UNIVERSIDADE DA CORUÑA

ESCUELA UNIVERSITARIA POLITÉCNICA

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO DE FIN DE GRADO

TFG Nº: **770G01A88**

TÍTULO: **CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO**

AUTOR: **Cristóbal García Camoira**

TUTOR: **Francisco Prieto Guerreiro**

FECHA: **JUNIO DE 2015**

Fdo.: EL AUTOR

Fdo.: EL TUTOR

TÍTULO: **CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO**

ÍNDICE GENERAL

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2015

AUTOR: EL ALUMNO

Fdo.: CRISTÓBAL GARCÍA CAMOIRA

Contenidos del TFG

Índice de figuras

Índice de tablas

TÍTULO: **CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO**

MEMORIA

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2015

AUTOR: EL ALUMNO

Fdo.: CRISTÓBAL GARCÍA CAMOIRA

Índice del documento MEMORIA

Capítulo 1

Objeto

El objeto principal del presente proyecto es la realización (diseño e implementación) de un control de posicionamiento de vehículos mediante la plataforma Arduino y un módulo “GPS/GPRS/GSM”, extrapolable a cualquier objeto en movimiento con el fin de hallar su posición.

Cabe destacar que el objetivo principal de la realización del presente TFG ha sido un fin didáctico, sobre todo para mí, autor de este trabajo, que siento curiosidad por saber como funciona el sistema de posicionamiento global, además de intentar conocer cómo controlar un dispositivo cualquiera, en este caso Arduino, mediante el envío de SMSs, posibilitando de esta forma el poder manejar cualquier dispositivo desde cualquier punto del mundo.

La implementación práctica a la que hemos llegado no pretende sustituir a los localizadores actualmente existentes, pues los más actuales utilizan la red 3G o 4G, permitiendo actualizar en Internet la situación de los dispositivos, rastrearlos (utilizando bases de datos) en lugar de enviar SMS.

Capítulo 2

Alcance

El presente trabajo ha sido realizado con la intención de ser implementado físicamente, con lo cual el alcance del mismo ha de cubrir todas las fases que procedemos a enumerar a continuación:

1. Fase de aprendizaje en la programación de Arduino y aplicaciones Android.
2. Fase de desarrollo e implementación del hardware y software asociado.
3. Fase de pruebas y comprobación del funcionamiento del dispositivo localizador.

En la primera fase, una de las más importantes de todo el proyecto, se dedica al aprendizaje de los diferentes lenguajes que se han utilizado para la elaboración del presente proyecto (lenguaje Arduino y lenguaje Android). En esta fase se han realizado e implementado físicamente en nuestra placa Arduino ejemplos muy básicos siguiendo múltiples tutoriales en internet.

En el caso del aprendizaje realizado para Android, se recomienda comenzar o tener conocimientos previos de programación en java, de no ser así, se puede comenzar directamente con programación en Android, realizando y tratando de comprender los ejemplos más simples que existen, como el que nos trae el programa “Andoid Studio” por defecto al crear un nuevo proyecto, el típico ejemplo de “Hola Mundo”, que no es más que mostrar en la pantalla de nuestro smartphone la frase típica que se muestra en todos los proyectos para el desarrollo de la primera aplicación. Intentaremos subir el nivel de dificultad paulatinamente hasta comprender el funcionamiento de una aplicación muy básica que incluye algunos objetos más, como por ejemplo un par de botones, y que al presionarlos muestren o quiten la frase “Hola Mundo” por pantalla.Así sucesivamente hasta que hacia el final del aprendizaje intentaremos introducir fragmentos de código de terceros (Google).

La segunda fase es la que le da forma al proyecto, el dispositivo debe incluir múltiples opciones de configuración , ser fácilmente transportable, opciones de selección del menú de forma intuitiva etc.

Todas estas opciones se incluyen en el capítulo (6). Además de cumplir con todos los requisitos, con el objeto de cumplir con que el dispositivo sea fácil de transportar, se realizará con

la herramienta “Solid Edge” una caja de plástico a medida, los planos referentes al la caja se presentan en los planos [10,11 y 12] esta caja se realizará con una impresora 3D, existente en la facultad gracias al departamento de ingeniería industrial.

En la tercera y última fase denominada como fase de pruebas es en la que nos encargaremos de que el prototipo realizado funcione de forma adecuada y de acuerdo a los parámetros que configuremos previamente en el menú, algunas de las pruebas que serán realizadas son las siguientes:

1. Comprobar que en el menú no exista ninguna errata de sintaxis.
2. Comprobar el funcionamiento de la herramienta física de selección, en este caso se utilizará un encoder rotativo.
3. Comprobar que el dispositivo localizador es capaz de enviar un SMS con el contenido de la hora y posición en que se encuentra.
4. Verificar si la posición enviada mediante SMS se sitúa en el lugar correcto, esto se realiza introduciendo en Google Maps las coordenadas enviadas por el dispositivo localizador teniendo en cuenta que las coordenadas solo nos indican los grados y los minutos (formato Dm “degrees, minutes”). Podremos utilizar el apartado de cálculos (capitulo 11) para convertir las coordenadas a un formato correcto (grados, minutos, segundos y dirección.
5. Verificar si la posición que obtiene Google Maps en nuestra aplicación de Android es la misma que en el apartado anterior.
6. Comprobar la comunicación entre nuestro smartphone y el dispositivo localizador, es decir, enviar al dispositivo localizador un comando mediante SMS y que este nos responda mediante SMS la posición en la que se encuentra.

Capítulo 3

Antecedentes

3.1. El sistema de posicionamiento GPS

El sistema de posicionamiento global (GPS) es un objeto que permite a una persona determinar en todo el mundo la posición de un objeto, una persona o un vehículo con una precisión hasta de centímetros (si se utiliza GPS diferencial), aunque lo habitual son unos pocos metros de precisión. El sistema fue desarrollado, instalado y empleado por el Departamento de Defensa de los Estados Unidos. El sistema GPS está constituido por 24 satélites y utiliza la trilateración (5) para determinar en todo el globo la posición con una precisión de más o menos metros.

El GPS funciona mediante una red de 24 satélites en órbita sobre el planeta tierra, a 20.200 km de altura, con trayectorias sincronizadas para cubrir toda la superficie de la Tierra. Cuando se desea determinar la posición, el receptor que se utiliza para ello localiza automáticamente como mínimo cuatro satélites de la red, de los que recibe unas señales indicando la identificación y la hora del reloj de cada uno de ellos. Con base en estas señales, el aparato sincroniza el reloj del GPS y calcula el tiempo que tardan en llegar las señales al equipo, y de tal modo mide la distancia al satélite mediante el método de trilateración inversa, la cual se basa en determinar la distancia de cada satélite respecto al punto de medición. Conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los satélites. Conociendo además las coordenadas o posición de cada uno de ellos por la señal que emiten, se obtiene la posición absoluta o coordenadas reales del punto de medición. También se consigue una exactitud extrema en el reloj del GPS, similar a la de los relojes atómicos que llevan a bordo cada uno de los satélites.

3.2. Control de posicionamiento vehículos

En el mercado actual existen multitud de localizadores GPS, de tamaños muy variados que se adaptan a la ergonomía de objetos, personas y animales.

Los localizadores GPS personales son ideales para realizar labores de control parental en menores, para la vigilancia a distancia de personas mayores o con problemas de Alzhei-

mer, excursionistas, etc. Algunos dispositivos pueden ser conectados a un servidor donde se podrá ver en tiempo real la posición del mismo.

Los localizadores GPS para animales son ideales para saber la posición de cualquier mascota.

Su tamaño puede llegar a ser realmente reducido, de hasta solo 50x46x17mm .Posee una correa ajustable para colocar fácilmente a cualquier tipo de mascota.

Solo con llamar al localizador GPS, este contestará con las coordenadas exactas que se podrán ver desde Google Maps, Google Earth, etc.

Este tipo de localizadores tienen una excelente precisión con un margen de error de tan solo 5 metros.

La mayoría de estos dispositivos utilizan servidores que almacenan la información de los usuarios, la empresa que los distribuye ofrece también servicios adicionales como por ejemplo aplicaciones web, estos servicios han de ser pagados mensualmente según las tarifas que imponga la propia empresa distribuidora, junto con el servicio de comunicación ofertado por la misma compañía o cualquier otra compañía de comunicaciones.

Cabe comentar que no existe constancia de dispositivos de localización que utilicen la plataforma de Arduino para su desarrollo.

Las ventajas del dispositivo que hemos desarrollado es que el único coste añadido y que se deberá abonar mensualmente es el de servicio de comunicación ofertado por cualquier compañía telefónica, la tarifa a elegir, independientemente de la compañía, ha de ser la que nos ofrezca los costes de SMSs lo más baratos posible.

El dispositivo que se pretende desarrollar en este proyecto está preparado para poder ser montado y utilizado, pudiendo utilizar una aplicación desarrollada para el mismo sin costes adicionales.

Capítulo 4

Normas y referencias

4.1. Disposiciones legales y normas aplicadas

En este apartado se contempla el conjunto de disposiciones legales, es decir, leyes y reglamentos de obligado cumplimiento que se han tenido en cuenta para la realización del presente documento. La normativa en la cual se basa este proyecto se puede subdividir en dos partes:

1. Normativa de la Escuela Universitaria Politécnica para la realización de trabajos de fin de grado.
    - Reglamento del trabajo de fin de grado para la Escuela Universitaria Politécnica de Ferrol.
    - UNE-EN-ISO 216.“Papel de escritura y ciertos tipos de impresos. Formatos acabados, series A y B”.
    - UNE 82100.“Magnitudes y unidades (partes de 0 a 13)”.
    - UNE 50132.“Numeración de las divisiones y subdivisiones en los documentos escritos”.
2. Normativas que afectan a la calidad y exigencias de fabricación de los componentes utilizados para su comercialización y uso en la Unión Europea.
    - DIRECTIVA 2004/108/CE del Parlamento Europeo y del Consejo de 15 de diciembre de 1004 relativa a la aproximación de las legislaciones de los Estados Miembros en materia de compatibilidad electromagnética.
    - UNE 20620-4:1980.“Materiales de base con recubrimiento metálico para circuitos impresos.Hoja de cobre”.
    - UNE 21302-521:2004.“Vocabulario electrotécnico. Capitulo 521: Dispositivos semiconductores y circuitos integrados”.
    - EN 123000/A1:1995.“Especificación genérica: Placas de circuitos impresos”.(Ratificada por AENOR en junio de 1996).

- EN 12300/A1:199.“Especificación intermedia:placas de circuitos impresos de simple y doble cara con agujeros no metalizadas”.(Ratificada por AENOR en junio de 1996).

4.2. Programas de cálculo

Los programas de cálculo que se han utilizado para la realización de este proyecto han sido prácticamente nulos dado que no se ha utilizado ningún programa potente para su realización dado que los instrumentos utilizados en el proyecto, nos facilitan mucho este proceso.

En este apartado también se incluirán el resto de programas que han sido claves para la realización del presente proyecto y que enumeraremos por orden de importancia.

1. Arduino IDE 1.0.5.
2. Android Studio.
3. Proteus Professional V8.1.
4. Hyperterminal.
5. Coolterm.
6. Autocad.
7. Solid Edge.
8. Microsoft Excel.

4.3. Otras referencias

En este apartado se incluyen otras referencias utilizadas para la realización del presente proyecto. En este caso la consulta de referencias a páginas de Internet supera la la consulta de libros realizada debido sobre todo a que la mayor parte de la información consultada es referida a los datasheets de los fabricantes de los diferentes dispositivos utilizados en el TFG (Trabajo de fin de grado).A continuación se elabora una lista de las referencias consultadas:

- `http://www.dfrobot.com/index.php?route=product/product&product_id=673`
- `http://playground.arduino.cc/es/es`
- `http://tallerarduino.com/category/videotutoriales/arduino-tutorials/`
- `http://www.dfrobot.com/image/data/TEL0051/3.0/SIM908_AT%20Command%20Manual_V1.01.pdf`

Capítulo 5

Definiciones y abreviaturas

En este apartado se incluyen todas las definiciones y abreviaturas que se utilizaron a lo largo del presente proyecto.

Por tanto, este capítulo no es más que de consulta de posibles palabras o abreviaturas que pueden no ser comprendidas.

- **USB:** Del inglés, Universal Serial Bus. Bus serie universal.
- **Sketch:** Entorno de programación en Arduino.
- **Jumper:** Conector minúsculo que hace posible a unión entre dos pines separados a cierta distancia.
- **EUP:** Escuela Universitaria Politécnica.
- **LCD:** Lyquid Crystal Display. Pantalla de cristal liquido.
- **TFG:** Trabajo de fin de grado.
- **Polling:** Operación de consulta constante, generalmente hacia un dispositivo de hardware, para crear una actividad sincrónica sin el uso de interrupciones, aunque también puede suceder lo mismo para recursos de software (p. ej. comprobar continuamente el estado de un pulsador).
- **I2C:** Inter-Integrated Circuit (Inter-Circuitos Integrados).
- **Hyperterminal:** Aplicación de Windows que le permite establecer una comunicación ordenador a ordenador o a cualquier otro dispositivo a través de una conexión telefónica convencional o por puerto serial.
- **Trilateración:** Es un método matemático para determinar las posiciones relativas de objetos usando la geometría de triángulos de forma análoga a la triangulación. A diferencia de ésta, que usa medidas de ángulo (junto con al menos una distancia conocida para calcular la localización del sujeto), la trilateración usa las localizaciones conocidas de

dos o más puntos de referencia, y la distancia medida entre el sujeto y cada punto de referencia. Para determinar de forma única y precisa la localización relativa de un punto en un plano bidimensional usando sólo trilateración, se necesitan generalmente al menos 3 puntos de referencia.

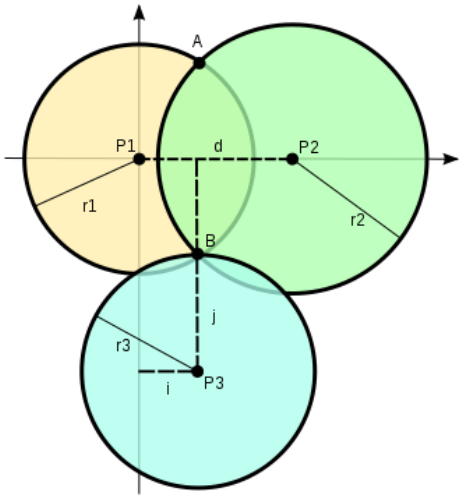


Figura 5.0.0.1 – Ejemplo de trilateración en un plano bidimensional

Estando en B, queremos conocer su posición relativa a los puntos de referencia P1, P2, y P3 en un plano bidimensional. Al medir r1 se reduce nuestra posición a una circunferencia. A continuación, midiendo r2, la reducimos a dos puntos, A y B. Una tercera medición, r3, nos devuelve nuestras coordenadas en B. Una cuarta medición también puede hacerse para reducir y estimar el error.

Si se desea conocer las expresiones matemáticas que sigue el sistema de trilateración se recomienda recurrir al siguiente enlace web:

`http://es.wikipedia.org/wiki/Trilateraci%C3%B3n`

- **Latitud:** Es la distancia angular que existe entre un punto cualquiera y el Ecuador.
- **Longitud:** Es la distancia angular que existe entre un punto cualquiera y el Meridiano de Greenwich.
- **Smartphone:** Es un tipo de teléfono móvil construido sobre una plataforma informática móvil, con una mayor capacidad de almacenar datos y realizar actividades, semejante a la de una minicomputadora, y con una mayor conectividad que un teléfono móvil convencional.

Capítulo 6

Requisitos de diseño

En este apartado se abarcarán los requisitos de diseño impuestos de forma previa a la realización del proyecto. Los requisitos que se imponen le afectan a las distintas partes del proyecto y, de alguna forma, concreta la forma de realización del presente trabajo en cuestiones de software, hardware y ergonomía. Estos se exponen a continuación:

1. Se empleara el software y el hardware de Arduino para la realización del dispositivo localizador.

2. En la medida de lo posible el software asociado se realizará de forma que pueda ser compatible con las diferentes placas de Arduino.

3. Para la recepción de datos referentes a la posición del dispositivo localizador y su posterior envío, se utilizará el módulo GPS/GPRS/GSM V3.0 disponible en el siguiente enlace:

   `http://www.dfrobot.com/index.php?route=product/product&product_id=673`.

4. El dispositivo ha de implementarse de forma que pueda ser transportado con facilidad y ser utilizado en cualquier lugar.

5. El dispositivo localizador ha de disponer de software asociado como por ejemplo una aplicación en Android instalada en un smartphone de los actuales que le permita situar con un marcador en un mapa la posición en la que se encuentra.

6. La aplicación a desarrollar ha de ser lo más simple posible para poder ser utilizada y entendida por cualquier tipo de usuario.

7. El dispositivo localizador debe ser capaz de comunicarse con cualquier dispositivo móvil seleccionado por el usuario mediante SMS cuyo contenido ha de ser el siguiente:

   - Hora
   - Latitud
   - Longitud

8. El dispositivo localizador ha de ser capaz de recibir órdenes desde cualquier dispositivo móvil que disponga de la aplicación realizada para la utilización del mismo.

9. El dispositivo localizador tiene que poder funcionar de forma autónoma, sin necesidad de conectarlo a un ordenador o a una fuente de alimentación.

10. En caso de conectar el dispositivo a un ordenador, se podrán visualizar a través de un terminal serie (hyperterminal)las tramas que recibe nuestro dispositivo localizador, así como los SMSs que es capaz de enviar a los diferentes terminales.

11. El dispositivo localizador debe contar con un LCD y un encoder con el que los usuarios puedan interaccionar con él de la forma más cómoda posible. Con estos dispositivos (LCD y encoder) se pretende que el usuario interaccione con el dispositivo localizador para, por ejemplo, seleccionar el número al cual el localizador le enviará mediante SMS la posición, cambiar el número de teléfono, seleccionar el tiempo de envío, cambiar la hora según el país.

    Estas y otras muchas opciones las podrá realizar el usuario mediante la interfaz implementada a través de estos dos dispositivos.

12. La disposición del menú del dispositivo localizador se realizara según los criterios que el diseñador crea más oportunos.

13. El dispositivo localizador ha de disponer de diversos modos de funcionamiento, es decir, poder elegir el momento del envío por SMS de la información adquirida por el dispositivo, que sea capaz de enviar mediante SMS cada cierto tiempo (configurado por el usuario) la información, o bien que simplemente la muestre por la pantalla que el dispositivo ha de disponer.

Capítulo 7

Análisis de las soluciones

Los requisitos de diseño impuestos en el capítulo [6] ya restringen bastante las soluciones a las que se pueden optar para la realización de este proyecto.

Uno de los requisitos menciona que el control de posicionamiento de vehículos se realice mediante la interfaz de Arduino, y no con cualquier otro micro-controlador, ya sea de la familia PIC de Microchip o Intel.

Existen en el mercado diferentes placas de Arduino que se adaptan a nuestras necesidades, algunas con un micro controlador más potente que otras, aunque es determinante el requisito de utilización del módulo GPS/GPRS/GSM V3.0 compatible con Arduino, con lo cual reducimos la elección de Arduino a tres posibles placas que podemos utilizar, dado que dicho módulo sólo puede encajar en Arduino Leonardo, Mega o Uno.

Otro de los requisitos fundamentales en el presente proyecto es el poder visualizar los datos que nos envía el módulo GPS en un LCD de 20 x 4 líneas y la transición entre los menús del programa debe ser controlada mediante un encoder rotativo.

Estos requisitos son fundamentales para saber el número de entradas y salidas que necesitamos y poder decidirnos entre cualquiera de los Arduinos disponibles en el mercado.

El número de recursos que necesitamos desde el punto de vista del micro controlador a utilizar, y sabiendo que el encoder dispone de tres pines y queremos utilizar el método de las interrupciones para gestionar su funcionamiento, además de conectarle un LCD de 20 x 4 líneas, que , preferiblemente manejaremos utilizando un módulo I2C,son:

1. Tres pines digitales de nuestro Arduino actuando como entradas dedicados a interrupciones.

2. Dos pines dedicados a la comunicación con nuestro LCD utilizando el método I2C (SDA y SCL).

3. Tres pines digitales de nuestro Arduino actuando como salidas que controlarán el módulo GPS.

4. Dos pines digitales de nuestro Arduino dedicados a la comunicación serie (Rx y TX).

En la siguiente figura (7.0.0.1) podemos ver las diferentes características que nos ofrecen las placas citadas.

							
Fabricante	Arduino	Arduino	Arduino	Arduino	Arduino	Arduino	Arduino
Modelo	Pro Mini	Nano	Uno	Mega / Mega 2560	Leonardo	Micro	Due
Microcontrolador	AVR Atmega 168 ó 328 8bits	AVR ATmega 168 ó 328 8bits	AVR ATmega 328 8bits	AVR ATmega2560 8bits	AVR ATmega 32u4 8bits	AVR ATmega 32u4 8bits	ARM SAM3X8E Cortex-M3 32bits
Frecuencia	16Mhz	16Mhz	16Mhz	16Mhz	16Mhz	16Mhz	84Mhz
Memoria RAM	2KiB	2KiB	2KiB	8KiB	2.5KiB	2.5KiB	96KiB (64+32KiB)
Memoria EEPROM	1KiB	1KiB	1KiB	4KiB	1KiB	1KiB	0
Memoria FLASH	16 ó 32KiB	16 ó 32KiB	32KiB	128 ó 256KiB	32KiB	32KiB	512KiB
Pines digitales entradas/salidas	14/14	14/14	14/14	54/54	20/20	20/20	54/54
Tensión/corriente pines digitales	3.3v ó 5v 40mA	5v 40mA	5v 40mA	5v 40mA	5v 40mA	5v 40mA	3.3v 3~15mA (130mA entre todos)
Pines analógicos entradas/salidas	6/0	8/0	6/0	16/0	12/0	12/0	12/2
Tensión/resolución pines analógicos	3.3v ó 5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	5v 10bits (1024 valores)	3.3v 12bits (4096 valores)
Pines con interrupción externa	2	2	2	6	2	2	-
Pines PWM	6	6	6	15	7	7	12
Conexiones Serial / UART	1	1	1	4	1	1	4
Conexiones I2C / TWI	1	1	1	1	1	1	2
Conexiones ISP / ICSP	1	1	1	1	1	1	1
Conexión USB	No (necesita adaptador externo)	Si	Si, USB-B	Si, USB-B	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB	Si, Nativa, MicroUSB
Conexión USB de depuración	No	No	No	No	No	No	Si, MicroUSB
Conexión Bluetooth	No	No	No	No	No	No	No
Conexión WiFi	No	No	No	No	No	No	No
Conexión Ethernet	No	No	No	No	No	No	No
Conexión USB Host	No	No	No	No	No	No	Si
Almacenamiento por SD	No	No	No	No	No	No	No
Corriente en el pin de 5v	-	500mA	500~800mA	500~800mA	500~800 mA	500mA	800mA
Corriente en el pin de 3.3v	-	50mA	50mA	50mA	50mA	50mA	800mA
Voltaje de alimentación por el USB	3.3v ó 5v (sin usb)	5v	5v	5v	5v	5v	5v
Voltaje de alimentación recomendado por el Jack	3.35 -12 V (modelo 3.3V) ó 5 - 12 V (modelo 5V)	7~12v	7~12v	7~12v	7~12v	7~12v	7~12v
Voltaje de alimentación limite por el Jack	-	6~20v	6~20v	6~20v	6~20v	6~20v	6~20v
Precio oficial	15+gi	-	20€+gi	40€+gi	18€+gi	18€+gi	39€+gi
Precio BBB	~4€	~9€	~10€	~12€	11€~	~16€	~38€

Figura 7.0.0.1 – Tabla comparativa de diferentes placas de Arduino

Las características que más nos interesan figuran en las dos siguientes figuras(7.0.0.2 y 7.0.0.3):

Interrupciones Arduino

Placa Arduino	Pin Digital De Cada Tipo Interrupción					
	Interrupción 0	Interrupción 1	Interrupción 2	Interrupción 3	Interrupción 4	Interrupción 5
UNO	2	3	-	-	-	-
DUE*	-	-	-	-	-	-
Leonardo	3	2	0	1	7	-
Mega	2	3	21	20	19	18
Micro	0	1	2	3	-	-
Mini	-	-	-	-	-	-
Nano	2	3	-	-	-	-
Ethernet	2	3	-	-	-	-
Esplora	-	-	-	-	-	-
Bluetooth	2	3	-	-	-	-
Fio	2	3	-	-	-	-
Pro Mini	22	3	-	-	-	-
Lilypad	-	-	-	-	-	-
*	La placa DUE permite establecer interrupciones en cualquier pin poniendo el nombre del mismo					

Figura 7.0.0.2 – Tabla de interrupciones en Arduino

Pines I2C Arduino

Board	I2C / TWI pins
Uno, Ethernet	A4 (SDA), A5 (SCL)
Mega2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

Figura 7.0.0.3 – Pines I2C Arduino

Ante las características que hemos citado anteriormente, podemos descartar Arduino UNO, dado que no dispone del número de interrupciones que necesitamos, aunque existe una alternativa que hemos llevado a la práctica para que funcione correctamente y el software realizado sea parcialmente compatible, esta alternativa la comentaremos al final de este capítulo.

Arduino Leonardo aparentemente cumple nuestras expectativas, dispone de 5 interrupciones y de comunicación I2C, pero existe un problema, en los pines 0 y 1 de Arduino se encuentra la comunicación serie, por lo tanto las interrupciones asociadas a esos dos pines tienen que dejarse de utilizar, en los pines 2 y 3 se encuentra la comunicación I2C, con lo cual son otras dos las interrupciones asociadas a esos dos pines las que no podemos utilizar, nos queda solo una para manejar el encoder, aunque necesitaríamos tres.

Llegados a este punto,y al haber comprado impulsivamente un Arduino Leonardo sin tener en cuenta la posible superposición de funciones que se pueden dar en un pin, tenemos que encontrar una solución ,esta es:

El LCD lo manejaremos utilizando el método de 4 bits, el control del LCD utilizaría en total 8 líneas que se desglosan de la siguiente forma:

- Dos líneas dedicadas a la alimentación.

- Cuatro líneas dedicadas al envío de los caracteres (4 bits).
- Dos líneas adicionales para el control de escritura en el LCD.

Utilizaremos pines digitales de nuestro Arduino configurados como salidas para el control de estas líneas del LCD, estos pines son el 8, 9, 10, 11, 12 y 13 y los pines que subministran la alimentación (5V y GND). El conexionado de estas líneas se puede observar en el plano [1] que, y en la sección [21.3](Conexionado de los circuitos). Dicho conexionado, aunque se realiza en Arduino MEGA, los pines utilizados tanto en Arduino Leonardo como en Arduino UNO, son los mismos (refiriéndonos al conexionado del LCD).

De esta forma los pines 2 y 3 de nuestro Arduino, asociados a las interrupciones 0 y 1 respectivamente (en UNO y en MEGA, en Leonardo los pines que controlan las interrupciones 0 y 1 son el 3 y el 2 respectivamente), quedarían libres, además del pin 7 (en Arduino MEGA utilizamos el pin 19), asociado a la interrupción 4, de esta forma podremos controlar por el método de las interrupciones, el manejo de la transición entre los menús con nuestro encoder rotativo.

Utilizaremos los pines 4, 5 y 6 de nuestro Arduino para el control del módulo GPS, el módulo GPS, por defecto, esta preparado para trabajar con los pines 3, 4 y 5 de nuestro Arduino, por ello se recomienda ver la sección 19.4 para conocer las modificaciones que se le han realizado.

Con todo lo dicho anteriormente, ya no disponemos de más entradas/salidas digitales.

Hablábamos anteriormente de una posibilidad o alternativa para la utilización de Arduino UNO en nuestro proyecto, el conexionado se realizaría de forma idéntica al que se hace con Arduino Leonardo, solo que, en lugar de utilizar tres interrupciones, utilizamos solo 2 , la 0 (pin2) y la 1 (pin 3) y el pin 7 de nuestro Arduino utilizará la técnica polling (capítulo 5) para comprobar si el pulsador de selección esta presionado.

El programa fuente puede hacerse compatible para Arduino Leonardo, y, con más dificultad, en Arduino UNO.

Realizando algunas modificaciones en el software que veremos en el capítulo [16](Compatibilidades del programa fuente con placas Arduino),podremos hacer compatible el código fuente realizado para Arduino Mega para un Arduino Leonardo, aunque de todas formas y sin conocer el motivo, el tiempo que tarda en posicionarse el módulo GPS utilizando Arduino Leonardo para la implementación del presente proyecto es mucho mayor que utilizando Arduino UNO o Arduino MEGA.

Con Arduino UNO también existe algún problema, en este caso parece ser un problema relacionado con la forma en la que esta dispuesto el código fuente, dado que, en teoría, los programas realizados para Arduino MEGA son perfectamente compatibles con Arduino UNO y viceversa (respetando la disposición de pines de Arduino UNO), por lo tanto será necesario comentar algunas lineas del programa fuente que se indica en el presente proyecto, estos cambios supondrían quedarnos sin el modo petición de usuario, con lo cual el desarrollo del presente proyecto con Arduino UNO nos limita una función muy importante.

A pesar de todas las molestias que nos hemos tomado para que este proyecto se pueda

realizar con las placas más básicas de Arduino con el fin de abaratar costes, el montaje final se ha realizado en Arduino Mega 2560, para poderle realizar pruebas adicionales.

Capítulo 8

Resultados finales

En el presente capítulo, se harán referencia a secciones presentes en la memoria, anexos, planos, estado de mediciones y presupuesto para abarcar el resultado final del presente proyecto.

En este capítulo se explicará de forma funcional apoyándonos en un diagrama de flujo el código fuente que hemos desarrollado para el dispositivo localizador, además de un ejemplo práctico de configuración del equipo y otro de utilización donde podremos observar los resultados finales obtenidos.

Se recomienda ver en el capítulo de planos (capitulo 17.2) todas las conexiones que se han realizado en Arduino (encoder rotativo, LCD y módulo GPS/GPRS/GSM), entonces podremos entender de forma más clara el código fuente implementado.

En el presente documento ya le hemos dedicado un capítulo bastante completo al software implementado en Arduino (capitulo 14)aunque en esta sección lo que se pretende es que el usuario comprenda la secuencia de operaciones que sigue el dispositivo para que funcione de forma exitosa, debido a este motivo, se harán referencias continuas al capítulo (14) para que pueda consultarse el código fuente.

8.1. Diagramas de flujo

8.1.1. Diagrama de flujo principal

A continuación se muestra el diagrama de flujo del dispositivo localizador.

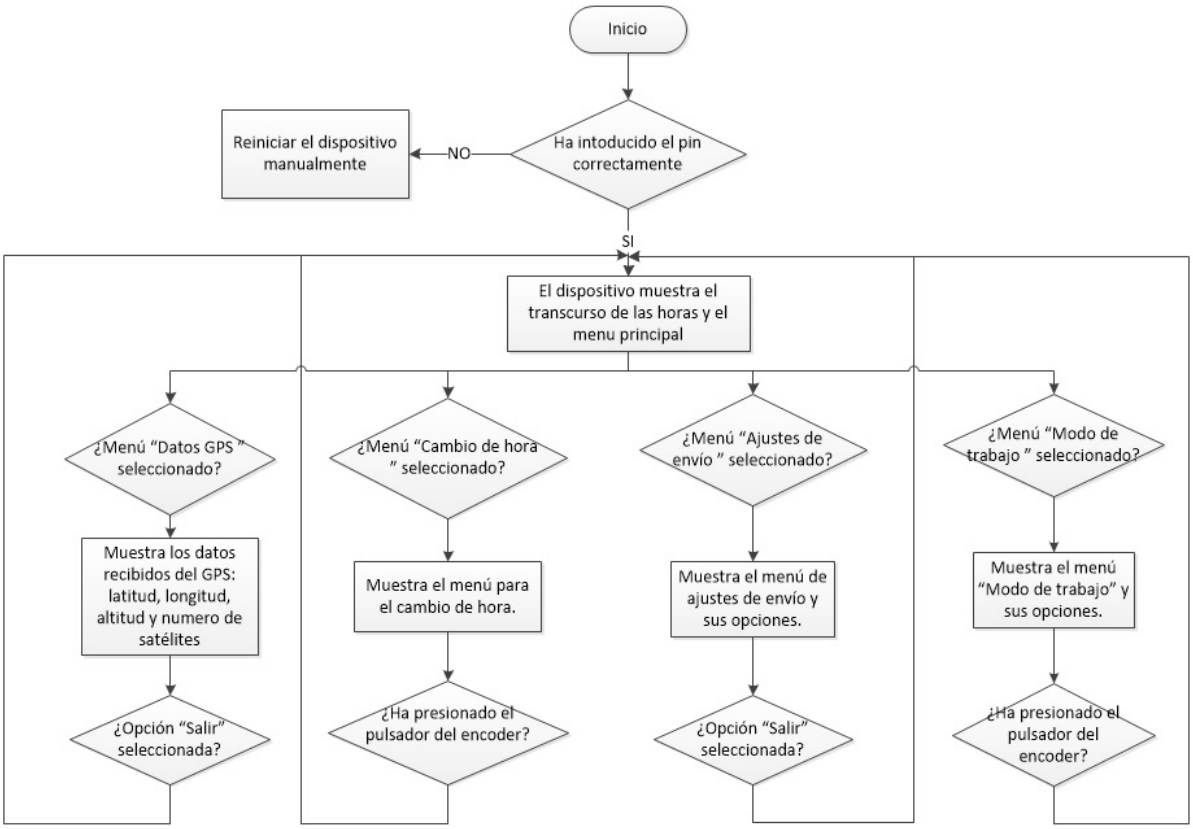


Figura 8.1.1.1 – Diagrama de flujo general del programa fuente

8.1.2. Diagrama de flujo del menú "Ajustes de envío"

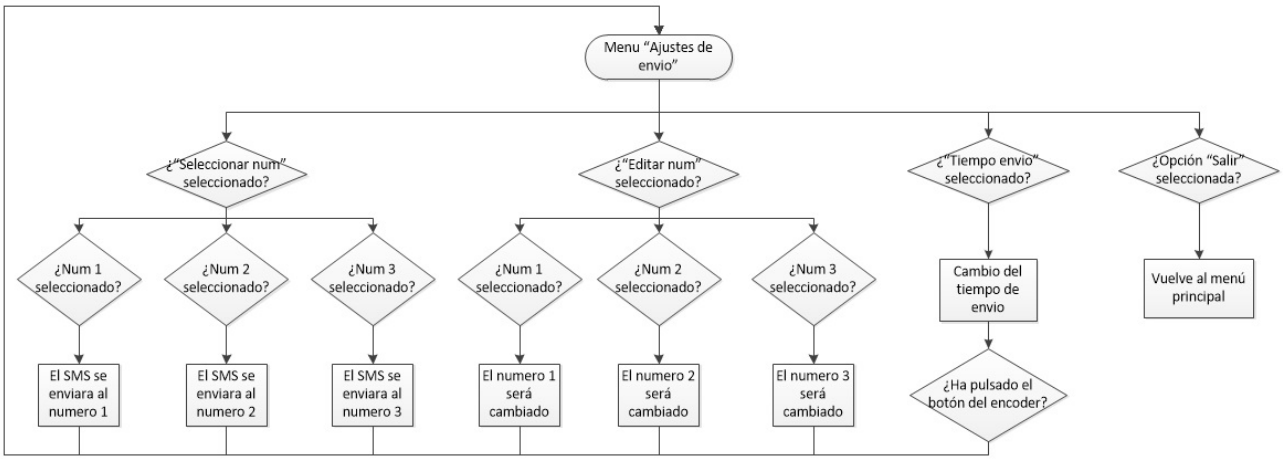


Figura 8.1.2.1 – Diagrama de flujo del menú "Ajustes de envío"

8.1.3. Diagrama de flujo del menú “Modo de trabajo”

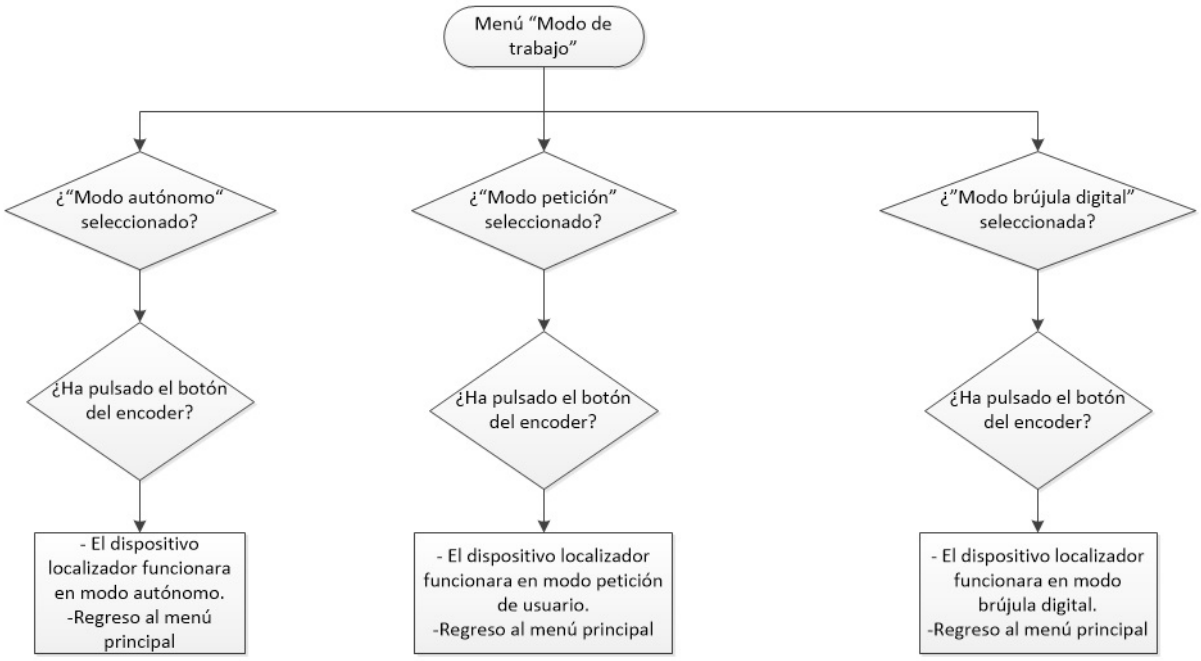


Figura 8.1.3.1 – Diagrama de flujo del menú “Modo de trabajo”

En la siguiente sección se explicarán de forma genérica las variables utilizadas durante la programación del dispositivo.

8.2. Librerías y variables

Si recurrimos al capítulo 14, en la sección 14.1 encontraremos las variables y librerías expuestas en esta sección, y, a pesar de que que ya han sido comentadas, se citan y se explican para una mayor claridad.

1. Incluimos librerías

   - **#include** $< stdio.h >$**:** Librería utilizada en la fase de pruebas, incluye la función printf.
   - **#include** $< stdlib.h >$**:** Librería que se puede utilizar para convertir, por ejemplo, un número en una cadena, en una variedad de bases (por ejemplo, decimal, binario).
   - **#include** $< LiquidCrystal.h >$**:** Librería utilizada para la visualización de caracteres en un LCD. En ella se gestiona la forma de comunicar nuestro Arduino con la pantalla LCD.
   - **#include** $< EEPROM.h >$**:** Librería utilizada para almacenar información en la memoria EEPROM de nuestro Arduino y no perderla en el caso de quedarnos sin batería.

2. Definición de pines

   - **LiquidCrystal lcd(11, NULL, 12, 13, 10, 9, 8):** Se definen los pines de Arduino que van a ser conectados a nuestro LCD de 20x4lineas.
   - #define TAMANO_ARRAY 150: Tamaño de buffer
   - #define habilitar_gps() digitalWrite (4, LOW)
   - #define deshabilitar_gps() digitalWrite (4, HIGH)
   - #define habilitar_gsm() digitalWrite (6, LOW)
   - #define deshabilitar_gsm() digitalWrite (6, HIGH)
   - enum PinAssignments
     encoderPinA = 2, Derecha (Llamado DT en nuestro encoder)
     encoderPinB = 3, Izquierda (Llamado CLK en nuestro encoder)
     clearButton = 19, switch (Llamado SW en nuestro encoder)
     ;

3. Variables de gestión del menú:

   - **encoderPos:** Contador para el encoder.
   - **lastReportedPos:** Variable de gestión para el cambio de posición del encoder. Almacena la posición anterior en la cual estaba la variable “encoderPos”.
   - **rotating:** Eliminación de rebotes.
   - **posicion:** Contador para el pulsador.
   - **lastPosicion:** Variable de gestión que detecta un flanco en el pulsador .Almacena la posición anterior en la cual estaba la variable “posicion”.

4. Variables para el servicio de la rutina de interrupción.

   - **A_set**
   - **B_set**
   - **C_set**
   - **actualizar**

5. Variables del programa:

   - **byteGPS:** Almacena cada byte que leamos del GPS para mandarlos al array.
   - **i:** Variable que utilizaremos como acumulador,inicializamos su valor en 1.
   - **c:** Variable que se incrementa a medida que introducimos un número de teléfono (p.ej).
   - **incremento:** Variable que suma o resta unidades al valor de las horas.

- **incremento_Aux:** Variable duplicada de “incremento” que suma o resta unidades al valor de las horas.
- **Estado:** Indica si ha encontrado o no la ubicación.
- **TramaGPG:** Array donde iremos almacenando los bytes leídos del GPS.
- ***pch:** Creamos un puntero de tipo char que apuntara al array de punteros (*GGA).
- ***GGA:** Array de punteros de tipo char.
- **sat:** Variable para almacenar el número de satélites.
- **menu:** Variable “menú” para la rotación del mismo.
- **numSel:** Variable “numSel” selecciona el número al cual enviar el SMS.
- **modoTrabajo:** Variable “modoTrabajo” indica el modo de funcionamiento. “modoTrabajo”=0 modo autónomo; “modoTrabajo”=1 petición de usuario ; “modoTrabajo”=2 brújula digital.
- **Flag_start:** Ayuda a comenzar el programa. Se pone a 1 cuando se introduce el pin, entonces comenzará el programa.
- **Array_Latitud:** Array donde se almacena la latitud del GPS.
- **Array_Latitud_sms:** Array donde se almacena la altitud del GPS para enviar el SMS.
- **Array_Longitud:** Array donde se almacena la longitud del GPS. **Array_Longitud_sms:** Array donde se almacena la longitud del GPS para enviar el SMS.
- **Array_Altitud:** Array donde se almacena la altitud del GPS.
- **Array_Hora:** Array donde se almacena la hora del GPS.
- **Array_HN:** Array donde se almacena la hora nueva del GPS.

6. Variables relacionadas con los números de teléfono.

- **numero_Tlf_1:** Array donde permanece el número 1 de teléfono.
- **numero_Tlf_1_Aux:** Array del primer número de teléfono al cual se le vuelcan las variables introducidas por el usuario.
- **numero_Tlf_2:** Array donde permanece el número 2 de teléfono.
- **numero_Tlf_2_Aux:** Array del segundo número de teléfono al cual se le vuelcan las variables introducidas por el usuario.
- **numero_Tlf_3:** Array donde permanece el número 3 de teléfono.
- **numero_Tlf_3_Aux:** Array del tercer número de teléfono al cual se le vuelcan las variables introducidas por el usuario.
- **Numero_PIN:** Array donde permanece el número PIN de la tarjeta SIM insertada en el módulo.
- **Numero_PIN_Aux:** Array del número Pin al cual se le vuelcan las variables que va introduciendo el usuario.

- **numero_buf:** Array que almacena el comando AT que establece el número de teléfono.
- **pin_buf:** Array que almacena el comando AT que establece el número PIN de la tarjeta SIM insertada en el módulo.
- **introduce_Num:** Almacena el número en modo char.
- **array_Minutos:** Array que almacena los minutos.

7. Variables que recogen un SMS recibido:

   - **answer**
   - **x**
   - **SMS:** Almacena toda la cadena del SMS recibido.
   - ***puntSms:** Creamos un puntero de tipo char.
   - ***arrayPuntSms:** Array de punteros de tipo char.
   - **telefonoSms:** Almacena el número de teléfono del SMS recibido.
   - **comillas:** Almacena las comillas de la segunda parte del array.
   - **fechaSms:** Almacena la fecha del SMS recibido.
   - **horaSms:** Almacena la hora del SMS recibido.
   - **smsRecibido:** Almacena el SMS recibido.

8. Variables de tiempo:

   - **Horas:** Guarda el valor entero de las horas.
   - **Horas_Aux:** Guarda el valor entero de las horas para el cambio de hora.
   - **Minutos:** Almacena el valor entero de los minutos.
   - **Minutos_Aux** Almacena el valor entero de los minutos para su comparación.
   - **Segundos:** Almacena el valor entero de los segundos.
   - **Segundos_Aux:** Almacena el valor entero de los segundos para su comparación.
   - **t_Envio:** Almacena el tiempo de envío.
   - **direccion:** Dirección inicial de la EEPROM donde guardaré datos.
   - **f:** Variable que se incrementa para guardar los teléfonos en EEPROM.

8.3. Configuración del LCD

Ene esta sección se muestra como se configura Arduino para que la utilización del LCD sea muy sencilla. Las librerías necesarias para ello ya se han indicado en la sección anterior.

Cabe comentar que la utilización de librerías, ya sea para un pantallas de cristal liquido (LCDs), sensores,motores,etc, nos facilitan enormemente la utilización de dichos componentes.

Para el LCD en cuestión, sólo se debe indicar el número de caracteres que posee cada línea (columnas) y el número de líneas que éste posee(filas). Esto se realiza en la función setup() del código fuente,(sección 14.2).

```
1 lcd.begin(20,4);
```

Lo anteriormente citado no funcionará si previamente no se define la variable LCD con los pines que va a utilizar nuestro Arduino para su uso.

```
1 LiquidCrystal lcd(11, NULL, 12, 13, 10, 9, 8);
```

Como aclaración del tema, se puede consultar el plano [1].

8.4. Configuración del dispositivo localizador

Los pasos para configurar correctamente el dispositivo se indican a continuación:

1. Introducir el PIN de la tarjeta insertada en el dispositivo localizador correctamente, de no ser así, deberemos apagar el módulo y volver a comenzar de nuevo.Ver sección (13.3).
2. Introducir un número de teléfono (podremos introducir hasta 3 números) al que se le enviará la información mediante SMS, este número se guardará en la memoria EEPROM de nuestro Arduino, de forma que cuando se gaste la batería, o se apague por cualquier motivo, no será necesario volver a introducirlo.Ver sección(13.4.3).
3. Seleccionar el número de teléfono al que deseamos enviar la información.Ver sección(13.4.3).
4. Seleccionar el tiempo de envío, es decir, la frecuencia con la que los SMSs serán enviados.Ver sección(13.4.3).
5. Seleccionar el modo de funcionamiento con el que deseamos trabajar, si deseamos que el dispositivo localizador nos envíe los mensajes automáticamente elegiremos el “Modo autónomo”, si deseamos que nos los envíe después de habérselo pedido previamente mediante el uso de la aplicación desarrollada en Android una vez que el módulo GPS encuentre la posición,elegiremos el modo “Petición usuario”,o bien si queremos que no nos envíe mensajes y utilizar el dispositivo a modo de brújula digital elegiremos el modo “Brújula digital”. Ver sección(13.4.4).
6. Configurar el huso horario correspondiente con el de nuestro país.Ver sección(13.4.2).

Nota:

- La segunda y la tercera opción solo son validas si configuramos el dispositivo tanto como modo “Petición usuario” o bien “Modo autónomo”, no sería necesario configurar estas opciones en el caso de que vayamos a trabajar en modo “Brújula digital”.
- La cuarta opción solo es necesario configurarla si vamos a utilizar el “Modo autónomo”.

- La última configuración que se realice será almacenada en la memoria de nuestro dispositivo, de forma que no será necesario configurarlo cada vez que se encienda.

8.5. Ejemplo de utilización

Como ejemplo de utilización vamos a suponer tres casos prácticos, cada uno, se corresponde con cada modo de configuración que posee el dispositivo.Empecemos:

1. Caso 1: Necesitamos conocer la posición relativa de un objeto en movimiento con una frecuencia determinada.
2. Caso 2: Necesitamos conocer la posición relativa de un objeto ya sea en movimiento o no, en un instante determinado.
3. Caso 3: No necesitamos conocer la posición relativa de ningún objeto, llevaremos el dispositivo con nosotros e iremos visualizando en la pantalla nuestra posición.

Para cada uno de los tres casos se recomienda un tipo de configuración del dispositivo distinta, este tipo de configuración la podemos ver en la sección anterior. En el primer caso se recomienda utilizar el “Modo autónomo”, en el segundo caso, el modo “Petición usuario” y en el tercer caso el modo “Brújula digital”.

8.6. Diagrama de flujo del funcionamiento del dispositivo en modo autónomo

Figura 8.6.0.2 – Diagrama de flujo del funcionamiento en modo autónomo

El diagrama de flujo que presenta la figura 8.6.0.2 se ha realizado para comprender el funcionamiento del dispositivo en modo autónomo y poder relacionar las funciones que se

presentan en el capítulo 14 con los hechos que en la figura se representan, tanto el envío del SMS desde Arduino como la recepción en nuestro terminal móvil y la representación de los datos recibidos en nuestra aplicación.

Comenzaremos enumerando las funciones más importantes que se utilizan durante el envío de un SMS desde Arduino, estas funciones se encuentran explicadas brevemente en su sección correspondiente.Estas funciones son:

1. **EnviaComandoAT** (ver sección 14.4.8)
2. **establecer_numero_tlf** (ver sección 14.4.7)
3. **envia_mensaje_gsm** (ver sección 14.4.6)
4. **finaliza_envio_gsm()** (ver sección 14.4.5)

Las funciones que realizan el tratamiento de los datos del SMS recibido en la aplicación realizada para nuestro smartphone están ubicadas dentro de la sección 15 del presente proyecto y estas son:

1. **onClickObtener** (ver sección 15.3)
2. **conversionGradosDecimal** (ver sección 15.3)
3. **obtenerMensaje** (ver sección 15.3)
4. **obtenerPosicion** (ver sección 15.3)

8.7. Diagrama de flujo del funcionamiento del dispositivo en modo petición de usuario

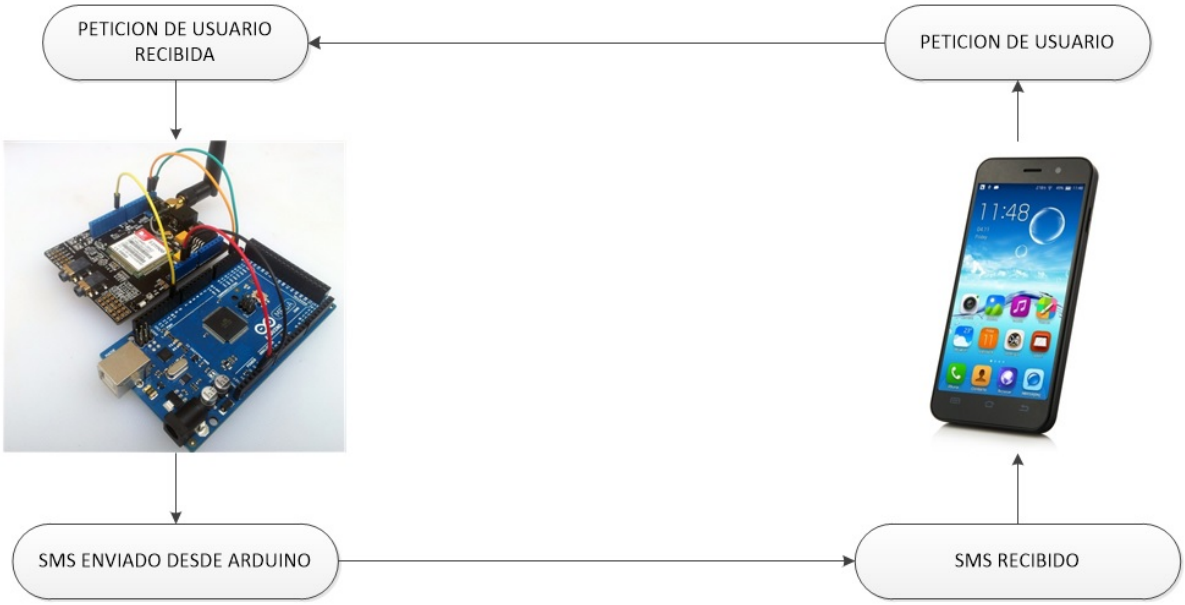


Figura 8.7.0.3 – Diagrama de flujo del funcionamiento en modo petición de usuario

El diagrama de flujo que presenta la figura 8.7.0.3 se ha realizado para comprender el funcionamiento del dispositivo en modo petición de usuario y poder relacionar las funciones que se presentan en el capítulo 14 con los hechos que en la figura se representan, tanto la petición de usuario realizada desde la aplicación de nuestro Smartphone, como la recepción de la misma en nuestro Arduino .

Comenzaremos enumerando las funciones más importantes que se utilizan durante el envío de la petición de usuario desde la aplicación realizada para nuestro Smartphone, estas funciones se encuentran explicadas brevemente en su sección correspondiente.Estas funciones son:

1. **sendSMS** (ver sección 15.3)

2. **onClickPedir** (ver sección 15.3)

Las funciones que realizan el tratamiento de los datos de la petición recibida en nuestro Arduino (que realmente es un SMS) están ubicadas dentro de la sección 15 del presente proyecto y estas son:

1. **descomponerSms()** (ver sección 14.4.4)

2. **leerSmsSim()** (ver sección 14.4.3)

3. **EnviaComandoAT** (ver sección 14.4.8)

   Tenemos que tener en cuenta en este caso que además de la intervención de las funciones citadas en este apartado, también intervienen las del apartado anterior, dado que en este modo Arduino , además de recibir la petición desde un smartphone, también envía la información hacia el mismo.

   El modo “Petición usuario” es el más completo y el que más requerimientos necesita de nuestro Arduino.

TÍTULO: **CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO**

ANEXOS

PETICIONARIO: **ESCOLA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2015

AUTOR: EL ALUMNO

Fdo.: CRISTÓBAL GARCÍA CAMOIRA

Índice del documento ANEXOS

Capítulo 9

Documentación de partida

En este anexo figura el justificante de asignación del trabajo de fin de grado donde se recoge el título del trabajo de fin de grado, el tutor, el número del TFG asignado y de forma breve los objetivos del proyecto.

Escuela Universitaria Politécnica Ferrol
UNIVERSIDADE DA CORUÑA

Intranet eup

ESCUELA UNIVERSITARIA POLITÉCNICA

ASIGNACIÓN DE TRABAJO FIN DE GRADO

En virtud de la solicitud efectuada por:
En virtud da solicitude efectuada por:

APELLIDOS, NOMBRE: *García Camoira,Cristóbal*
APELIDOS E NOME:

DNI: [redacted] **Fecha de Solicitud:** FEB2015
DNI: *Fecha de Solicitude:*

Alumno de esta escuela en la titulación de Grado en Ingeniería en Electrónica Industrial y Automática, se le comunica que la Comisión de Proyectos ha decidido asignarle el siguiente Trabajo Fin de Grado:

O alumno de esta escola na titulación de Grado en Enxeñería en Electrónica Industrial e Automática, comunícaselle que a Comisión de Proxectos ha decidido asignarlle o seguinte Traballo Fin de Grado:

Título T.F.G:Control de posicionamiento de vehiculos mediante Arduino

Número TFG: *770G01A88*

TUTOR:(Titor) *Prieto Guerreiro,Francisco*

COTUTOR/CODIRECTOR:

La descripción y objetivos del Trabajo son los que figuran en el reverso de este documento*:*

A descrición e obxectivos do proxecto son os que figuran no reverso deste documento.

Ferrol a Jueves, 5 de Marzo del 2015

Retirei o meu Traballo Fin de Grado o dia _____ de _____ do ano _____

Fdo: García Camoira,Cristóbal

DESCRIPCIÓN Y OBJETIVO:El objeto principal del proyecto es el desarrollo y construcción de un sistema hardware y el software asociado que permita llevar a cabo el posicionamiento del sistema mediante el uso de la plataforma arduino y un modulo GPS/GPRS/GSM, de forma que nos permita conocer en cualquier instante la posicion en la que esté situado el vehículo en donde va integrado.

El sistema será capaz de enviar informacion a otros dispositivos a traves de diferentes redes de comunicaciones que permita situarlo en una coordenadas concretas en un mapa, obtener la trayectoria seguida en una ruta, etc.

También, deberá permitir recibir comandos de configuración y control desde cualquier dispositivo remoto asociado.

Capítulo 10

Comandos AT utilizados

Los comandos AT, también conocidos como comandos Hayes (en honor a su desarrollador Dennis Hayes), son una serie de instrucciones que conforman un interfaz de comunicación entre usuario y módem. Su abreviatura AT por la que son mundialmente conocidos estos comandos proviene de la palabra 'attention'.

Aunque la finalidad principal de los comandos AT fue la comunicación con módems, la telefonía móvil GSM/GPRS también adoptó este lenguaje como estándar de comunicación.

En la actualidad, todos los terminales móviles GSM poseen una serie específica de comandos AT que nos permiten configurarlos por medio de estas instrucciones e indicarles una serie de acciones que queremos que ejecuten, tales como marcar un número de teléfono, enviar o leer un SMS, consultar el estado de conexión a la red, leer o escribir en la agenda de contactos, etc.

Gracias a que la transmisión de comandos AT no depende del canal de comunicación a través del cual estos sean enviados (cable, infrarrojos, Bluetooth, etc.), podremos utilizar nuestra placa Arduino para transmitir dichos comandos a un módulo GPRS/GSM que sea capaz de interpretarlos y actuar en consecuencia. Como son muchos los comandos existentes (véase el manual de comandos AT oficial de nuestro módulo GPS/GPRS/GSM(SIM908) , únicamente vamos a mencionar junto con una breve descripción, aquellos que puedan resultarnos de principal interés en el desarrollo del proyecto:

AT: Con este comando verificaremos que el módulo está recibiendo nuestras instrucciones. Si todo es correcto, tras enviarlo debe responder con un "OK". En caso contrario no obtendremos respuesta alguna.

AT+CPIN?: Utilizaremos esta instrucción para comprobar si nuestra tarjeta SIM está desbloqueada o si por el contrario, requiere introducir el código PIN para proceder con el desbloqueo de la misma. Cuando la SIM esté operativa responderá con un "ready".

AT+CPIN="**":** En el caso de que necesitemos introducir el PIN, éste es el comando que debemos enviar, escribiendo los 4 dígitos correspondientes al código de desbloqueo en el lugar de los asteriscos, delimitado entre comillas.

AT+CREG?: Con este comando estamos preguntando por el estado de conexión a la red. La respuesta recibida seguirá la siguiente notación: +CREG: <n>,<stat>, donde:

- stat = 0 : No registrado, no está buscando una conexión de red
- stat = 1 : Registrado a la red nacional
- stat = 2 : No registrado, pero buscando la red
- stat = 3 : Registro denegado
- stat = 5 : Registro tipo roaming

AT+COPS?: Mediante esta instrucción recibiremos la confirmación de la compañía en la que está registrada nuestra tarjeta SIM (Movistar, Orange, Vodafone, etc.)

AT+CMGF=<f>: Selecciona el formato del mensaje SMS, donde:

- f = 0 : modo PDU
- f = 1 : modo texto

AT+CSCS="IRA": Con este comando seleccionamos el set de caracteres que es utilizado para el envío de SMS. En nuestro caso nos interesa configurar el módulo en modo "IRA" (International Reference Alphabet).

AT+CMGS="6XXXXXXXX": A través de este comando marcaremos el número de móvil al que queremos hacer llegar el SMS. Irá delimitado por comillas.

AT+CMGR=<r>: Es el comando utilizado para leer SMS almacenados en la memoria de la tarjeta SIM. En lugar de <r> pondremos el número correspondiente a la posición del mensaje que queremos leer. Por ejemplo, si queremos leer el último mensaje recibido, pondremos un "1", si queremos leer el penúltimo mensaje pondremos un "2", y así sucesivamente.

AT+CPMS="SM": Con esta instrucción seleccionamos el directorio del que queremos leer un mensaje de texto. En función de las siglas que pongamos tras el igual, podemos recopilar la información de distintas memorias. En este caso, con las siglas "SM", seleccionamos como directorio la memoria de la tarjeta SIM.

AT+CMGD=<n>: Este comando sirve para eliminar SMS de la memoria de la tarjeta SIM. Sustituiremos <n> por la posición en la memoria que ocupe el SMS que queremos borrar. Por ejemplo, para borrar el primer mensaje almacenado en la memoria de la SIM la instrucción sería AT+CMGD=1.

AT+CMGL="ALL": Con este comando podemos ver en una lista todos los SMS que tengamos en la SIM, tanto leídos, como no leídos.

+CMTI: "SM", <pos> : Esta es la instrucción que recibiremos automáticamente por parte del módulo cuando se reciba un SMS, donde <pos> es el número correspondiente a la posición en memoria en la que se ha almacenado dicho mensaje.

Por ejemplo, si tenemos la memoria de la SIM vacía y nos llega un SMS, la instrucción que nos enviaría el módulo sería +CMTI: "SM", 1.

NOTA: Para el envío de cualquiera de los comandos AT a través de Hyperterminal o del entorno de desarrollo de Arduino, debemos seleccionar siempre los caracteres fin de línea y retorno de carro.

Capítulo 11

Cálculos

Para la realización del presente trabajo, los cálculos realizados se reducen prácticamente a la conversión de grados decimales a grados sexagesimales para la representación en un mapa proporcionado por Google, las coordenadas enviadas por el dispositivo localizador.

Haciendo uso de las recomendaciones de los fabricantes de los diferentes dispositivos utilizados para la elaboración de nuestro localizador, display LCD, encoder, hemos reducido los cálculos considerablemente.

11.1. Notación sexagesimal

Podemos expresar una cantidad en grados, minutos y segundos, las partes de grado inferiores al segundo se expresan como parte decimal de segundo, ejemplo:

- $12°34'34''$
- $13°3'23,8''$
- $124°45'34,70''$
- $-2°34'10''$

Teniendo cuidado como norma de notación, no dejar espacio entre las cifras, es decir: escribir $12°34'34''$ y no $12°34'34''$. Podemos también representar en forma decimal la medida de un ángulo en representación sexagesimal teniendo en cuenta que:

- $1' = (1/60)° = 0,01666667°$ (redondeando a ocho dígitos)
- $1'' = (1/60)' = (1/3600)° = 0,00027778°$
- Así $12°15'23'' = 12° + 15(1/60)° + 23(1/3600)° = 12,25639°$

11.2. Notación decimal

Una cantidad en grados se puede expresar en forma decimal, separando la parte entera de la fraccionaria con la coma decimal, se divide en 60 en la forma normal de expresar cantidades decimales, lo que se busca es transformar en minuto y el segundo números decimales, por ejemplo:

- $23,2345°$
- $12,32°$
- $-50,265°$
- $123,696°$

11.3. Ejemplo de conversión

Nuestro dispositivo localizador nos ha enviado un SMS indicándonos la posición en la cual se encuentra.

- La longitud es la siguiente: 08°20,423613'W
- La latitud es la siguiente: 43°17,610714'N

Los datos que nos envía el dispositivo son números con notación decimal hasta los minutos, dado que por defecto nuestro localizador nos envía las coordenadas en formato DM (degrees minutes), con lo cual la conversión la tenemos bastante sencilla.

Necesitamos saber cuáles son las coordenadas sexagesimales tanto de la longitud como de la latitud a partir de los minutos.

Comenzamos con la longitud.

Sabemos cuales son los grados (08) y los minutos (20), nos queda por saber cuáles son los segundos, para ello debemos realizar la siguiente operación.

Sabemos que los minutos son 20,423613.

Restamos los minutos y el resultado lo multiplicamos por 60 para obtener los segundos:

- $20,423613 - 20 = 0,423613$
- $0,423613 * 60 = 25,41678$

Ya sé que son 25"

Luego obtenemos que la longitud es con exactitud 08°20'25,41678".

Continuamos con la latitud:

Sabemos cuales son los grados (43) y los minutos (17), nos queda por saber cuáles son los segundos, para ello debemos realizar la siguiente operación.

Sabemos que los minutos son 17,610714.

Restamos los minutos y el resultado lo multiplicamos por 60 para obtener los segundos:

- $17,610714 - 17 = 0,610714$
- $0,610714 * 60 = 36,64284$

Ya sé que son 36”

Luego obtenemos que la longitud es con exactitud 43°17’36,64284”.

Nos queda por averiguar el significado de las letras N (norte) y W (oeste, en inglés, west) que se observan al final del SMS, estas nos indican el cuadrante del hemisferio en el que estamos, la figura 11.3.0.1 nos aclara de forma gráfica lo anteriormente citado.

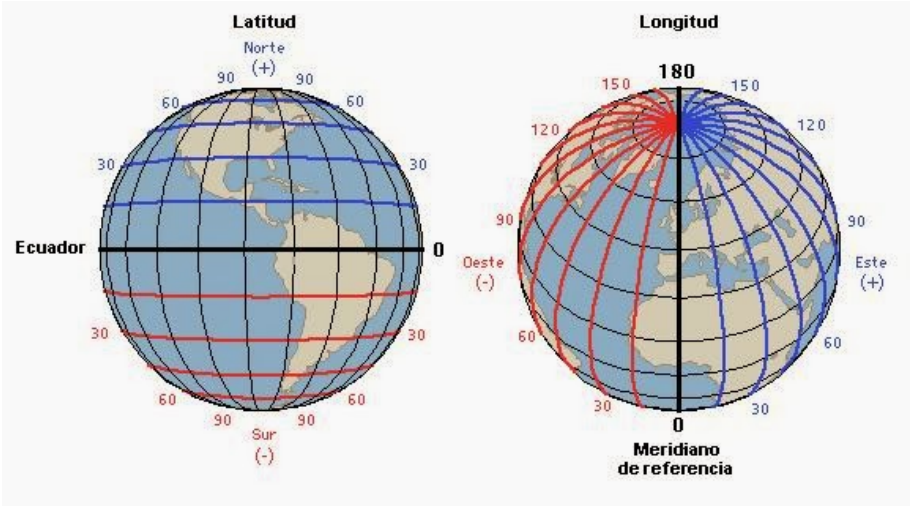


Figura 11.3.0.1 – Aclaración del sistema de posicionamiento mediante coordenadas

Capítulo 12

Comprobación de resultados

En el presente capítulo se comprueban los resultados obtenidos de nuestro dispositivo localizador. Se realizarán las pruebas en sus diversos modos de funcionamiento y comprobaremos que la posición enviada desde un punto se corresponde con la posición real, además de comprobar que el envío lo hace de forma correcta y al número de teléfono que se ha indicado durante la configuración.

Hemos realizado las pruebas en los tres modos de funcionamiento de los cuales se disponen,“Modo autónomo”, “Petición de usuario” y modo “ Brújula digital”.

En modo autónomo hemos comprobado que los SMSs se envían con la frecuencia necesaria e impuesta durante la configuración del dispositivo, y hemos verificado la concordancia del SMS enviado con el que hemos recibido en nuestro terminal móvil, siendo esta correcta.

En modo brújula digital hemos comprobado que la posición varía en la pantalla a medida que nos movemos por el territorio,una comprobación sencilla dado que la comprobación más completa es la que se realiza en el modo “Petición de usuario”.

En modo “Petición de usuario”se ha realizado una prueba a distancia, siendo el dispositivo localizador el que se encontraba lejos del usuario, concretamente en el laboratorio de electrónica I de la EUP en situada en el campus de Serantes, Ferrol.

Para la realización de la prueba ha colaborado un compañero el cual ha tenido que instalar la aplicación elaborada en Android para el manejo del dispositivo.

Una vez que mi compañero ha instalado la aplicación, alimentamos el dispositivo localizador con la tensión y corriente adecuadas (ver sección 21.4), y, una vez conectada, tendremos que seguir los pasos que se nos indican en el manual de usuario (capitulo 13),que básicamente son los siguientes:

1. Introducimos el PIN de la tarjeta insertada en nuestro dispositivo localizador.

2. Configuramos los parámetros necesarios para que el destinatario (mi compañero) reciba el SMS enviado desde nuestro dispositivo localizador después de que envíe la petición.

   El apartado de configuración del dispositivo localizador de encuentra en la sección 8.4.

3. Una vez configurados los parámetros necesarios, es necesario esperar a que el dispositivo localizador encuentre la posición en la que se sitúa. En el caso de que mi compañero envíe una petición para obtener la posición, el dispositivo localizador no responderá hasta que encuentre la posición.

4. Una vez encontrada la posición, cuya imagen del dispositivo localizador se muestra en la figura 12.0.0.1 el dispositivo procederá a notificarla mediante un SMS enviado al terminal móvil de nuestro compañero, que previamente ha sido configurado.

   NOTA: El dispositivo localizador, en modo “Petición de usuario” solo enviará la posición si el dispositivo ha recibido la petición por parte del usuario.

5. Si el dispositivo se encuentra encendido y con la posición localizada el tiempo máximo que tardará en enviar el SMS será de aproximadamente 1 minuto.

6. El formato del SMS recibido es el que se muestra en la figura 12.0.0.2 , comparando la figura 12.0.0.1 con la figura 12.0.0.2, podremos ver un pequeño error , esto es debido a que la imagen de la figura 12.0.0.1 ha sido obtenida minutos después de que el dispositivo localizador mandase el SMS, le ha dado de tiempo a concretar un poco más su posición , dado que ha tomado la referencia de más satélites de lo que lo había hecho cuando envió el primer SMS.

7. Una vez recibido el SMS mi compañero, conociendo la posición en la cual se encuentra el dispositivo localizador, comprueba, pulsando el botón “Obtener posición” de la aplicación que la posición que le indica es la correcta. La posición indicada se muestra en la figura 12.0.0.3.

8. Luego de haber recibido la posición en nuestro Smartphone, y a pesar de haber verificado que la posición era correcta, comprobamos en Google Maps de la web, si la posición que nos ha enviado el dispositivo localizador mediante SMS coincide con la posición que nos marca nuestra aplicación, esto lo corroboramos visualizando la figura 12.0.0.4.

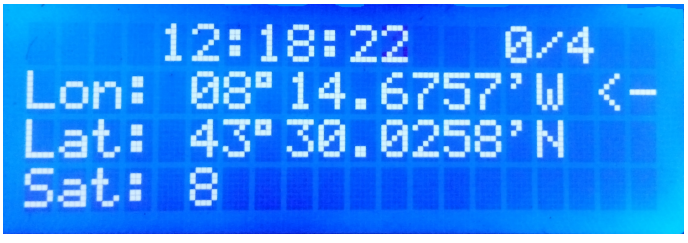


Figura 12.0.0.1 – Posición enviada desde el localizador hacia el smartphone

Figura 12.0.0.2 – Mensaje recibido del dispositivo localizador

Figura 12.0.0.3 – Posición de nuestro dispositivo en nuestra aplicación

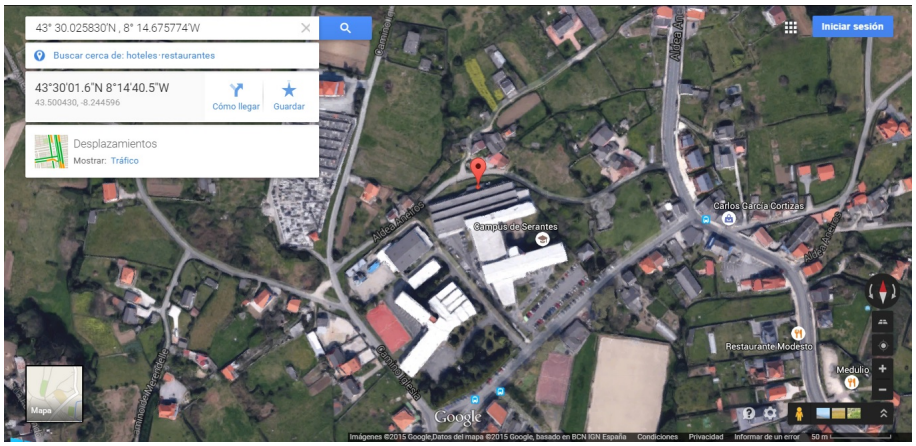


Figura 12.0.0.4 – Posición de nuestro dispositivo en Google Maps

Capítulo 13

Manual de usuario y especificaciones

En este capítulo se desarrolla el manual de usuario para la correcta utilización del dispositivo. En primer lugar se indican las instrucciones de uso paso a paso para poder utilizar el localizador GPS y por otra parte se detallan las especificaciones técnicas del producto.

Este capítulo servirá de guía para el uso del dispositivo, donde se muestran las diferentes pantallas que ayudan a configurar el dispositivo al gusto del usuario. También nos indica (al final de este capítulo) las especificaciones técnicas del localizador GPS.

13.1. Puesta en marcha del equipo

El primer paso antes de la utilización del dispositivo su correcta alimentación. La alimentación del equipo puede ser obtenida a través del puerto USB de nuestro ordenador,ocurre un problema, y es que, en el mejor de los casos, cada puerto es capaz de subministrar una corriente de aproximadamente 500mA a una tensión de 5V, esto es suficiente si no deseamos enviar ningún SMS, entonces el dispositivo lo estaremos utilizando como una especie de brújula digital, por otro lado no siempre debemos de disponer de nuestro ordenador de sobremesa o portátil cada vez que deseemos poner en marcha nuestro localizador.

La forma más correcta y funcional de alimentar nuestro dispositivo es alimentar nuestro Arduino con una fuente que nos subministre una tensión de 9V y 2000mA de corriente, la solución comercial más económica es la utilización de una batería lipo de 7,4V y 2200mAh.

Cabe comentar que la alimentación externa, ya sea subministrada por una batería lipo o una fuente de alimentación lineal, conmutada... es compatible con la que se subministra por el cable USB cuando al menos se subministra por ella 300mA, es decir, las dos alimentaciones pueden estar conectadas simultáneamente, encargándose el Arduino de seleccionar automáticamente la alimentación procedente de la fuente externa. De esta forma podremos utilizar la comunicación serie (USB) para ver las tramas en el PC que nuestro localizador envía al dispositivo móvil que tengamos seleccionado en ese momento.

Para más información acerca de la alimentación del dispositivo, se recomienda observar la sección (21.4).

El segundo paso a realizar es verificar que nuestra tarjeta SIM está insertada en el módulo

GPS/GPRS/GSM V3.0 en el espacio que indica la figura 7.0.0.2 designado como SIM CARD SOCKET.

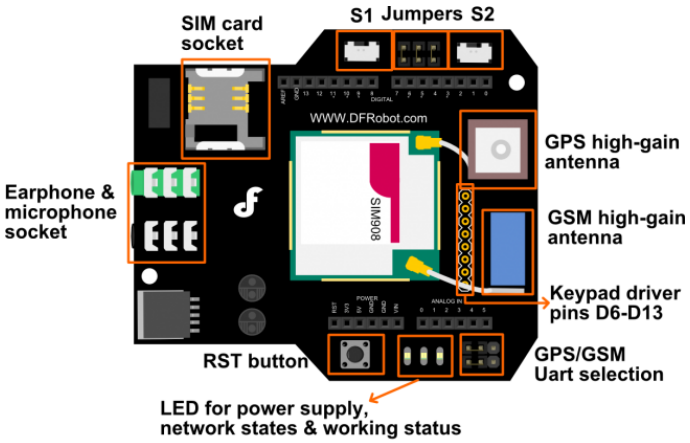


Figura 13.1.0.1 – Introducción de la tarjeta SIM

Una vez resueltos los dos pasos anteriores ya tenemos preparado nuestro dispositivo para su correcto funcionamiento

13.2. Viajando por el menú de usuario

La transición de los menús es muy sencilla, para seleccionar la opción deseada dentro del menú de usuario, rotaremos hacia la derecha o hacia la izquierda la rueda de nuestro encoder rotativo, de esta forma la flecha de selección de la opción bajará o subirá respectivamente.

Para efectuar la selección presionaremos hacia adentro (en la dirección de la pantalla) la rueda el encoder rotativo, de esta forma la opción deseada quedará seleccionada.

13.3. Configuración del código PIN

Una vez alimentado el dispositivo y haber encendido el interruptor, el dispositivo localizador nos muestra la pantalla que muestra la figura 13.3.0.2, una pantalla de bienvenida que nos da dos opciones, la primera es la de usar el pin nuevo y la segunda es la de usar un pin antiguo.

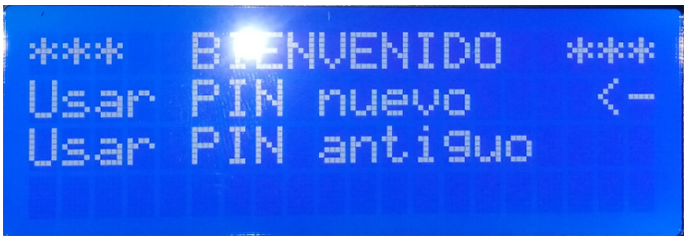


Figura 13.3.0.2 – Pantalla de bienvenida

13.3.1. Usando un pin nuevo

Si hemos escogido esta opción, nos aparecerá algo en el LCD similar a lo que nos muestra la figura 7.0.0.3. Esta opción ha de ser seleccionada obligatoriamente si utilizamos una tarjeta SIM diferente a la que anteriormente estábamos utilizando o bien durante el primer uso del dispositivo.

Hacemos girar la rueda del encoder hacia la derecha o hacia la izquierda según donde se encuentre el número a seleccionar, un guión vertical nos señalará el número. Una vez señalado el número, para hacer efectiva la selección, presionaremos hacia adentro la rueda del encoder rotativo.

Si nos hemos equivocado durante la introducción de un número que no sea el último, podremos borrarlo seleccionando la letra ”B”, pues seleccionándola, se borra el número anterior introducido.

El número PIN de nuestra tarjeta se compone de cuatro números que solo el usuario del dispositivo debe conocer, a no ser que lo desee utilizar más de una persona.

Una vez introducidos los cuatro números nos aparecerá una pantalla como muestra la figura 13.3.1.2, indicando que el número se ha introducido correctamente.

Aunque el número introducido sea un número incorrecto, se mostrará el mismo mensaje, dado que no existe la comprobación de si el número pin es correcto dentro del módulo, con lo cual, en el momento en el que el módulo desee enviarnos un SMS, no podrá hacerlo, debido a que utiliza el PIN incorrecto para el envío del SMS, por lo tanto **SE RUEGA INTRODUCIR EL NÚMERO PIN CORRECTAMENTE** para el correcto funcionamiento del módulo.

De no ser así, deberemos resetear el dispositivo e introducir el PIN correctamente.

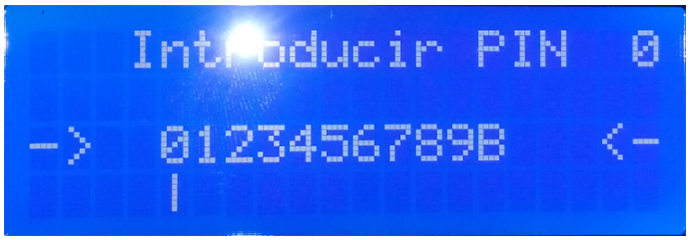


Figura 13.3.1.1 – Configuración del código PIN

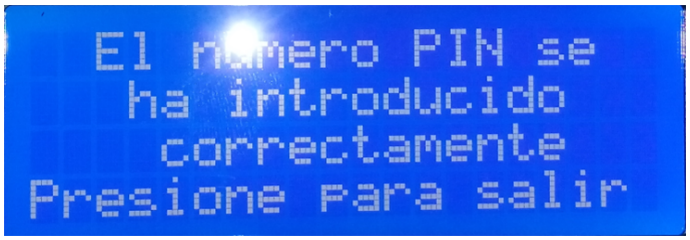


Figura 13.3.1.2 – Introducción del número PIN correcta

13.3.2. Usando un pin antiguo

Si se ha escogido esta opción, el programa utilizará el PIN guardado anteriormente. Se recomienda escoger esta opción siempre que utilicemos la misma tarjeta SIM, para el uso del dispositivo localizador. Una vez seleccionada esta opción el programa nos llevará al menú principal , del cual hablaremos posteriormente.

13.4. Menú principal

El presente menú se muestra en las figuras 13.4.0.1 y 13.4.0.2, este menú nos permite el acceso a los datos enviados por nuestro módulo GPS (Datos GPS), podremos ajustar la hora dependiendo del país en el que nos situemos (Ajuste hora), seleccionar el número de teléfono al cual se le enviará el SMS, modificar el mismo y cambiar el tiempo de envío del mensaje (Ajustes de envío), y seleccionar si deseamos que el dispositivo localizador nos envíe los mensajes automáticamente , nos los envíe después de habérselo pedido previamente mediante la aplicación de Android una vez que el módulo GPS encuentre la posición, o bien no nos los envíe (Modo de trabajo).

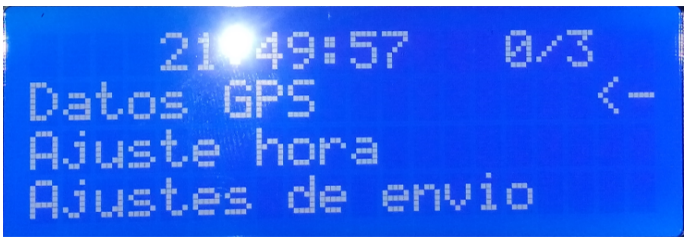


Figura 13.4.0.1 – Pantalla del menú principal 1

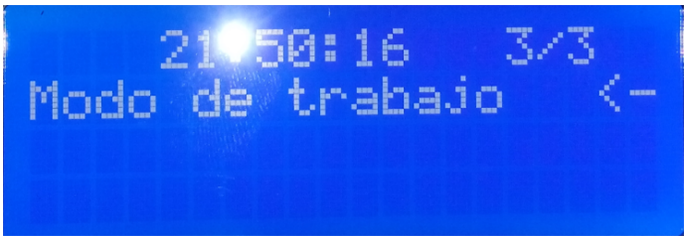


Figura 13.4.0.2 – Pantalla del menú principal 2

13.4.1. Menú Datos GPS

Este es el menú más sencillo, solo se encarga de mostrar por pantalla los datos que obtiene el módulo GPS, Longitud, Latitud, el número de satélites y la altitud de la zona, la figura 13.4.1.1 muestra lo que se refleja por pantalla cuando el módulo aun no ha obtenido la posición, una vez obtenida el módulo mostrará por pantalla en formato Gm la posición donde se encuentra el dispositivo , tal y como se muestra en la figura 13.4.1.2.

Figura 13.4.1.1 – Pantalla del dispositivo GPS buscando posición

Figura 13.4.1.2 – Pantalla del dispositivo GPS con posición encontrada

Aunque no se muestra en las figuras anteriores, la pantalla LCD también nos muestra la altitud en la cual esta situado nuestro dispositivo localizador así como la opción de salir, que una vez seleccionada, se retorna al menú principal.

13.4.2. Menú Ajuste hora

Este menú sirve para el cambio de hora del dispositivo, la hora cambiada será la que el dispositivo envíe una vez que localice la ubicación.

El menú de cambio de hora se muestra e la figura 13.4.2.1.

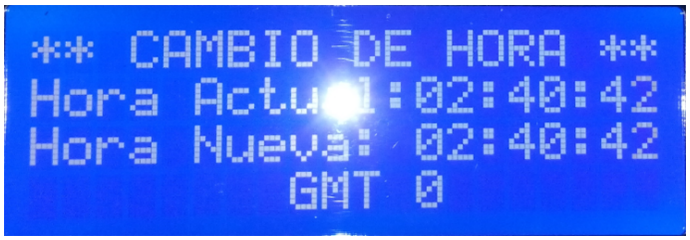


Figura 13.4.2.1 – Pantalla del menú de cambio de hora

La hora que se muestra por defecto es la existente en el ecuador (UTC+0), en España sería necesario ajustar la hora de este dispositivo dos veces al año , ya que disponemos del horario de invierno (UTC+1) y el horario de verano (UTC+2).

Como dato orientativo en la figura 13.4.2.2 se muestran las Zonas horarias europeas , donde podemos ver el huso horario que utiliza cada país en Europa.

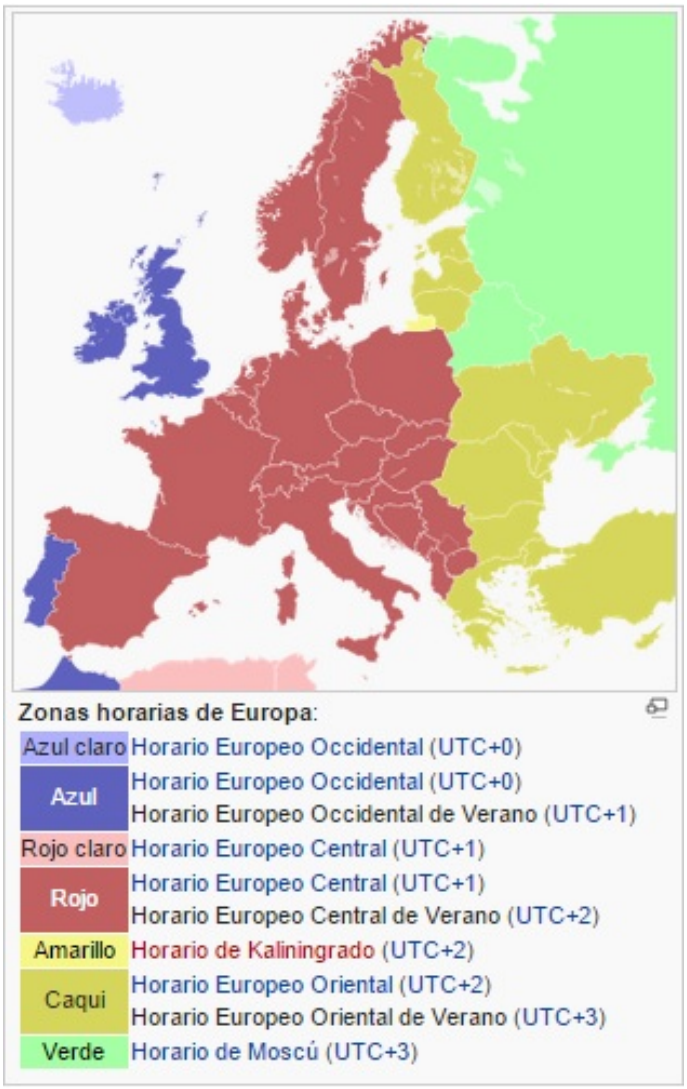


Figura 13.4.2.2 – Zonas horarias europeas

El cambio de hora de nuestro dispositivo permite su uso en países con un huso horario de entre (UTC-11) hasta (UTC+12), estas son las diferencias máximas y mínimas que existen a nivel mundial en cuanto a los husos horarios, de todas formas , existen países como la India, Sri Lanka, cuyo huso horario es de (UTC + 5:30) , las Islas Cocos, Myanmar (UTC + 6:30) o también Nueva Zelanda (Islas Chatham) con un huso horario de (UTC +12:45) , en los cuales esa media hora o bien los 45 minutos de más no los tenemos en cuenta, por lo tanto con la utilización del dispositivo localizador en esos países tendríamos un error en la hora, esto es debido a que inicialmente nuestro dispositivo localizador estaba pensado para trabajar en zona europea, aunque dependiendo del éxito del presente producto se le harán las mejoras pertinentes.

Tenemos que tener mucho cuidado también a la hora de cambiar de país dado que el envío de SMS de un país a otro nos puede salir muy caro debido al cambio de operadora, el ejemplo sería comunicarnos desde nuestro localizador GPS con otro terminal móvil haciendo uso de la cobertura de otra compañía, con lo cual a nuestra tarifa habitual se le suman esos costes

adicionales, generalmente muy costosos.

Atendiendo a lo anteriormente citado, se recomienda cambiar de compañía telefónica para hacer uso del dispositivo localizador en el momento de cambiar de país.

Una vez seleccionado el huso horario de nuestro país, presionamos sobre la rueda del encoder rotativo, y el display LCD nos mostrará algo similar a lo que aparece en la figura 13.4.2.3, de esta forma confirmaremos el cambio de hora.

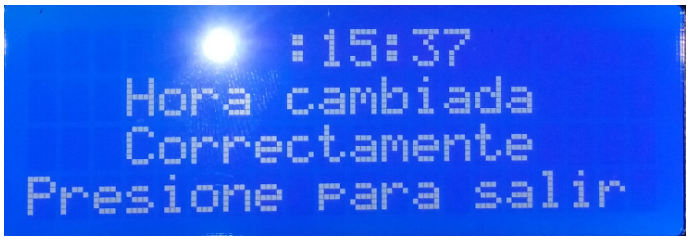


Figura 13.4.2.3 – Pantalla de confirmación del cambio de hora

13.4.3. Menú Ajustes de envío

Este es el menú más extenso y en el cual tendremos varias opciones, opciones que se muestran en la figura :

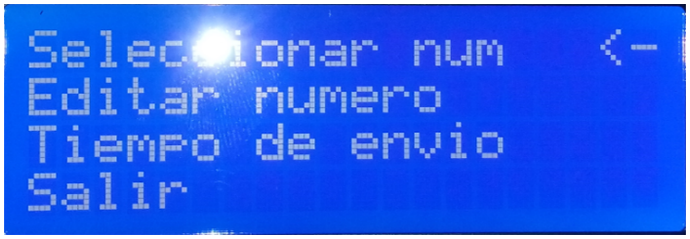


Figura 13.4.3.1 – Pantalla del menú Ajustes de envío

A continuación se detallan las opciones del menú Ajustes de envío:

- **Seleccionar num:** Eligiendo esta opción podremos seleccionar el número al cual le deseamos enviar la información desde nuestro dispositivo localizador, podremos elegir entre tres números diferentes. Si seleccionamos “Num 1”, los SMS que enviará nuestro dispositivo localizador, los enviará al “Num 1” seleccionado, esta pantalla se muestra en la figura 13.4.3.2.

  Si en esta opción, no se visualiza ningún número, esto quiere decir que todavía no se le ha introducido ningún número, por tanto es necesario acudir al menú “Editar número” e introducirle el número de teléfono en el cual recibiremos el SMS.

  Una vez que se ha seleccionado el número, nuestro display nos mostrará en pantalla lo que se muestra en la figura 13.4.3.3 indicando que el número que ha sido seleccionado,y presionando de nuevo la rueda del encoder hacia adentro volveremos de nuevo al menú de ajustes de envío.

Todos los números de teléfono tienen que haber sido introducidos previamente en la opción “Editar número” para poderse almacenar en la memoria del dispositivo.

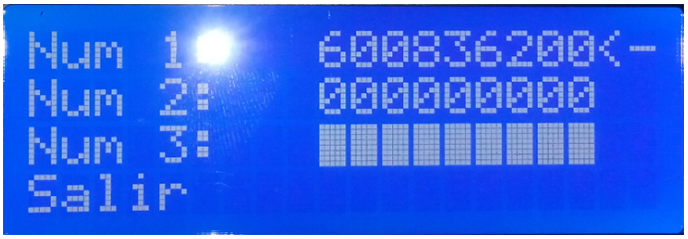


Figura 13.4.3.2 – Pantalla de selección de número

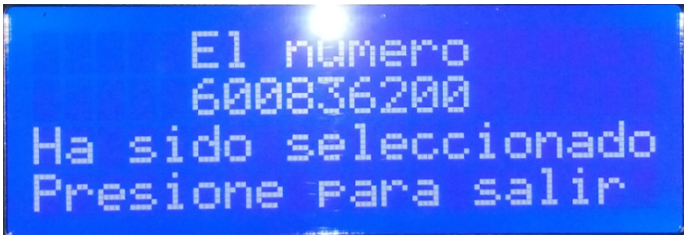


Figura 13.4.3.3 – Pantalla del número que ha sido seleccionado

- **Editar número:**Eligiendo esta opción, podremos cambiar el número de de teléfono introducido previamente o bien introducir uno nuevo por primera vez. El número se introduce de forma idéntica a la que se introduce el número PIN.

  La forma de seleccionar el número a cambiar es idéntica a la que se hizo en el menú “Seleccionar num” tal y como se muestra en la imagen 13.4.3.2.

  Una vez seleccionado el número a cambiar hacemos girar la rueda del encoder hacia la derecha o hacia la izquierda según donde se encuentre el número a seleccionar, un guion vertical nos señalará el número. Una vez señalado el número, para hacer efectiva la selección, presionaremos hacia adentro la rueda del encoder rotativo.

  Si nos hemos equivocado durante la introducción de un número que no sea el último, podremos borrarlo seleccionando la letra ”B”, pues seleccionándola, se borra el número anterior introducido.

  Una vez que hallamos seleccionado las nueve cifras que componen un número de teléfono, automáticamente el LCD nos mostrará que el número ha sido cambiado correctamente tal y como muestra la figura 13.4.3.4, presionando la rueda del encoder hacia adentro el número habrá sido cambiado y se retornará al menú ajustes de envío nuevamente.

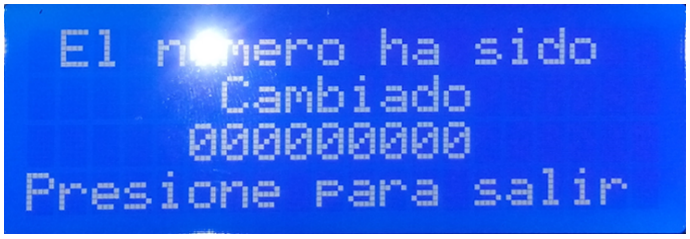


Figura 13.4.3.4 – Pantalla del número que ha sido cambiado

- **Tiempo de envío:**Esta opción solo será útil si la opción modo autónomo del dispositivo localizador esta activada. El tiempo de envío es el intervalo de tiempo que transcurre desde un envío de un SMS hasta el siguiente.

  Lo anteriormente dicho no es exactamente así, el SMS será enviado si el minuto en el que estamos es múltiplo del tiempo de envío fijado, también es lo mismo que decir que el punto de partida se toma siempre desde el minuto cero independientemente de que el usuario elija un tiempo de envío en un momento determinado,dado que si por ejemplo estamos en el minuto 7 y escogemos un envío cada 3 minutos el siguiente envío se efectuará en el minuto 9, eso si, el envío solo se efectuará si el dispositivo localizador ha obtenido la ubicación .

  El usuario podrá escoger el tiempo de envío de los SMS desde 1 minuto hasta 59 minutos, aunque a partir de un tiempo de envío de 31 minutos es indiferente ya que el número de envíos durante esa hora va a ser únicamente uno.

  La pantalla de cambio del tiempo de envío se muestra en la figura 13.4.3.5, una vez seleccionado el tiempo de envío rotando la rueda del encoder rotativo, presionamos hacia adentro la misma, y el tiempo de envío queda seleccionado.

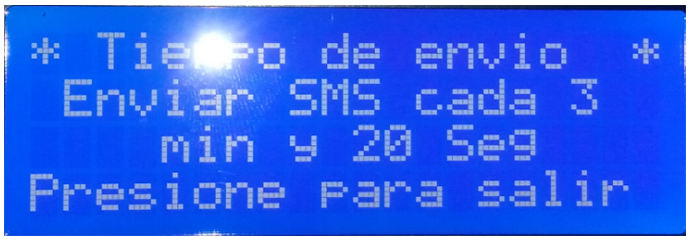


Figura 13.4.3.5 – Pantalla de cambio del tiempo de envío

13.4.4. Menú Modo de trabajo

Como ya hemos citado anteriormente, en esta opción elegiremos si deseamos que el dispositivo localizador nos envíe los mensajes automáticamente ,deseamos que nos los envíe después de habérselo pedido previamente mediante el uso de la aplicación desarrollada en Android una vez que el módulo GPS encuentre la posición, o bien no nos envíe mensajes y utilizar el dispositivo a modo de brújula digital.

Una vez seleccionado el menú “Modo de trabajo”,que se muestra en la figura 13.4.0.2, visualizaremos en el display lo que muestra la figura 13.4.4.1, y, dependiendo de nuestra elección nos mostrará la figura 13.4.4.2 ,la figura 13.4.4.3 o bien, la figura 13.4.4.4 dependiendo de si hemos seleccionado el “Modo autónomo” ,el modo “Petición de usuario” o bien el modo“Brújula digital” respectivamente.

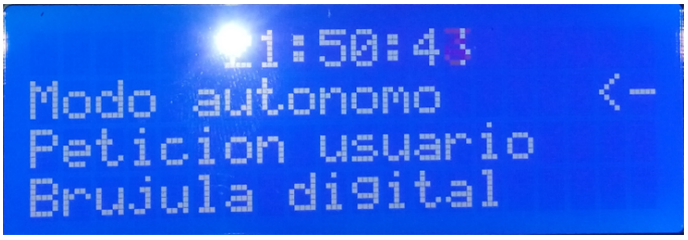


Figura 13.4.4.1 – Pantalla de selección de los modos de funcionamiento

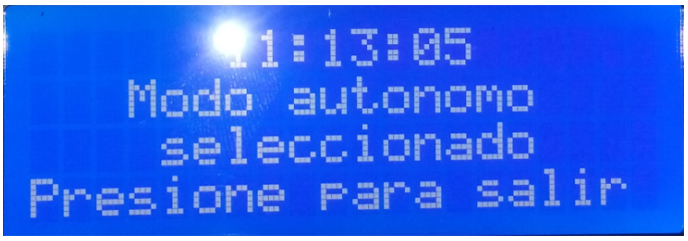


Figura 13.4.4.2 – Pantalla de selección del “Modo autónomo”

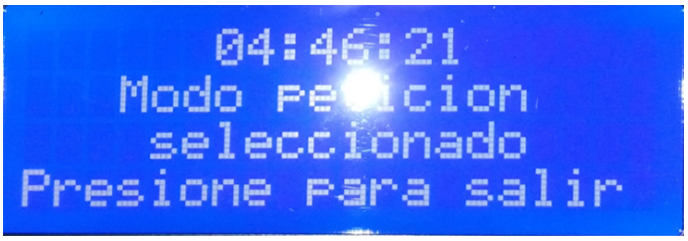


Figura 13.4.4.3 – Pantalla de selección del modo “Petición de usuario”

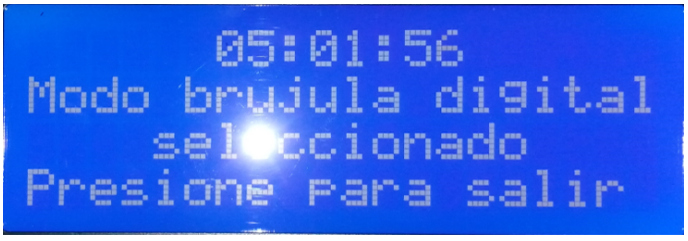


Figura 13.4.4.4 – Pantalla de selección del modo “Brújula digital”

Una vez seleccionado el modo de funcionamiento y habiendo presionado la rueda del encoder retornaremos de nuevo al menú “Ajustes de envío”.

13.5. Utilización de la aplicación Android

La utilización de la aplicación desarrollada es muy simple, una vez que vayamos recibiendo los SMSs procedentes del módulo podremos presionar el botón “Obtener posición”, en el caso de que el dispositivo localizador esté configurado en “Modo autónomo”, justo en ese instante se mostrará en el mapa proporcionado por Google un marcador indicando la situación del dispositivo.

Si presionamos encima del marcador nos mostrará también la hora de localización.

En el caso de que nuestro dispositivo esté configurado en modo “Petición de usuario”, no recibiremos SMSs cada intervalo de tiempo, somos nosotros los que le tendremos que enviar una petición al aparato para que este nos responda con la información pedida, por tanto tendremos que pulsar el botón “Pedir posición” y esperar (en el peor de los casos alrededor de 2 minutos) a que el módulo nos responda con un SMS conteniendo la posición en la cual se sitúa el módulo, una vez recibido el SMS pulsaremos de nuevo el botón “Obtener posición” y ocurrirá lo anteriormente citado (se mostrará en el mapa proporcionado por Google un marcador indicando la situación del dispositivo) .

Figura 13.5.0.5 – Aplicación Android para el control del dispositivo localizador

13.6. Especificaciones del equipo

A continuación se enumeran todas las especificaciones generales, eléctricas, térmicas y técnicas del dispositivo.

13.6.1. Especificaciones generales

1. Posibilidad de la elección del modo de funcionamiento del dispositivo.
   - Funcionamiento autónomo.
   - Funcionamiento a petición de usuario.
   - Funcionamiento a modo de brújula digital.
2. Ajuste del huso horario según el país.
3. Envío de SMSs cuyo contenido es la hora, latitud y longitud.
4. Envío de los SMSs a un solo número de teléfono de los existentes en memoria
5. Almacenamiento interno de hasta 3 números de teléfono , al cual les enviará los SMSs.
6. Posibilidad de seleccionar el número de teléfono al cual se le desea enviar el SMS de entre los tres posibles números existentes en memoria.
7. Cambio de los números de teléfono.
8. Tiempo de envío de SMS configurable.
9. Posibilidad de visualización de las tramas recibidas y los comandos enviados al módulo GPS con un ordenador a través de la interfaz serial de Arduino o con el Hyperterminal del ordenador.
10. Posibilidad de control a distancia del dispositivo localizador mediante la utilización de la aplicación Android diseñada para el mismo.
11. Visualización de la posición en un mapa mediante dicha aplicación Android diseñada para el mismo.

13.6.2. Especificaciones eléctricas y térmicas del equipo

1. **Alimentación:** Batería lipo 7,4V 2200mAh.
2. **Consumo medio:** Depende del uso.Se pueden producir picos de consumo durante el envío de SMSs de 16W.
3. **Rango de temperatura para su funcionamiento:** 0ºC – 50ºC.

13.6.3. Especificaciones técnicas

1. **Tiempo de respuesta del dispositivo localizador:** Alrededor de 1 minuto como máximo si el dispositivo está encendido y se ha ubicado correctamente.
2. **Error de precisión** 10m.

3. **Banda GSM:** Cuatribanda.
4. **Voz:** No.
5. Batería interna recargable y reemplazable 2200 mAh y (7.4V).
6. **Memoria Interna:** Si.
7. **Método de comunicación:** GSM.
8. **Posición por:** Tiempo y Distancia.
9. **Ahorro de Energía:** No.
10. **Antenas:** GPS, GSM.
11. **Cubierta:** Plástica.
12. **Fabricado en:** España.

Capítulo 14

Código fuente Arduino

En este anexo se presenta todo el código fuente que debe ser grabado en Arduino MEGA 2560.

14.1. Librerías y variables

```
1  //************************************************************************
2  // GPS/GPRS/GSM Module V3.0
3  //************************************************************************
4  // Autor : CRISTOBAL GARCIA CAMOIRA
5  //************************************************************************
6  // # Pasos:
7  // # 1. Ponemos el interruptor S1 en el modo prog (A la derecha).
8  // # 2. Ponemos el interruptor S2 en el modo Arduino(A la izquierda).
9  // # 3. Ponemos el interruptor UART en el centro de sus tres posibles posiciones.
10 // # 4. Subir el programa al Arduino.
11 // # 5. Poner el interruptor s1 hacia la izquierda en el modo comm.
12 // # 6. Resetear el aparato.(RST)
13 // # 7. Si se obtienen valores de tipo 'inf', ve afuera y espera a que la recepción
       de datos sea mejor (mejor cobertura).
14 // # wiki link- http://www.dfrobot.com/wiki/index.php/GPS/GPRS/GSM_Module_V3.0_(SKU:
       TEL0051)
15 /*
16 GPS + Display 4bits
17 En este proyecto aprendemos a conectar un módulo GPS y
18 a tratar la información recibida para mostrar
19 en un display la latitud, longitud, hora y altitud.
20 */
21
22 //Incluimos librerías
23 // #include <Wire.h>                 // La necesita la librería LiquidCrystal_I2C.h
24 // #include <LiquidCrystal_I2C.h>    // Para usar un display I2C
25  #include <stdio.h>
26  #include <stdlib.h>
27  #include <LiquidCrystal.h>
```

```
28 #include <EEPROM.h>
29
30 LiquidCrystal lcd(11, NULL, 12, 13, 10, 9, 8);
31
32 // LiquidCrystal_I2C lcd(0x27,20,4);
33 // Establece la dirección de memoria 0x27 para un display de 20 caracteres y 4
       líneas.
34
35 #define TAMANO_ARRAY 150 // Tamaño de buffer
36
37 #define habilitar_gps()       digitalWrite (4, LOW)
38 #define deshabilitar_gps()    digitalWrite (4, HIGH)
39
40 #define habilitar_gsm()       digitalWrite (6, LOW)
41 #define deshabilitar_gsm()    digitalWrite (6, HIGH)
42  /*
43  Rutina para nuestro menú con encoder rotativo.
44  El codificador rotativo genérico tiene tres pines, visto desde el frente: ACB
45  Giro a la derecha A (encendido) -> B (encendido) -> A (apagado) -> B (apagado)
46  Contador CW rotación B (encendido) -> A (encendido) -> B (apagado) -> A (apagado)
47  Puede también tener 2 pines a mayores siendo estos los correspondientes a un
        interruptor Conectado en este caso al pin 19 de nuestro Arduino.
48
49  Normalmente los encoders rotativos de 3 pines, tienen el pin de masa en el medio
        de los tres.
50  */
51  enum PinAssignments
52  {
53  encoderPinA = 2,    // Derecha (Llamado DT en nuestro encoder)
54  encoderPinB = 3,    // Izquierda (Llamado CLK en nuestro encoder)
55  clearButton = 19,   // switch (Llamado SW en nuestro encoder)
56  /* Conectar +5v y Gnd en los pines restantes.*/
57  };
58
59  volatile int encoderPos=0;                  // Contador para el dial.
60  volatile unsigned int lastReportedPos = 1;  // Gestión para el cambio.
61  static boolean rotating=false;              // Eliminación de rebotes.
62  volatile unsigned int posicion = 0;         // Contador para el pulsador.
63  volatile unsigned int lastPosicion = 1;     // Gestión para el cambio.
64
65  /* Variables para el servicio de la rutina de interrupción.*/
66
67  boolean A_set = false;
68  boolean B_set = false;
69  boolean C_set = false;
70  boolean actualizar = false;
71
72  /* Variables del programa*/
73
```

```
74  byte byteGPS = 0;              // Almacena cada byte que leamos del GPS para
        mandarlos al array.
75  int i = 1;                     // Variable que utilizaremos como acumulador,
        inicializamos su valor en 1.
76  int c=0;                       // Variable que se incrementa a medida que
        introducimos un número de teléfono (p.ej).
77  volatile int incremento;       // Variable que suma o resta unidades al valor de
        las horas.
78  volatile int incremento_Aux;   // Variable duplicada de incremento que suma o
        resta unidades al valor de las horas.
79  int Estado = 0;                // Guarda el estado de la conexión.
80  char TramaGPG[TAMANO_ARRAY];   // Array donde iremos almacenando los bytes leídos
        del GPS.
81  char *pch;                     // Creamos un puntero de tipo char.
82  char *GGA[15];                 // y un array de punteros de tipo char.
83  int sat = 0;                   // Variable para almacenar el número de satélites.
84  int menu=-2;                   // Variable menú para la rotación del mismo.
85  int numSel=1;                  // Variable numSel selecciona el número al cual
        enviar el SMS.
86  int modoTrabajo=0;             // Variable modoTrabajo indica el modo de
        funcionamiento. 0 modo autónomo; =1 petición de usuario ; =2 brujula digital.
87  boolean Flag_start = false;    // Ayuda a comenzar el programa.
88
89  char Array_Latitud[14]={' ',' ','\337',' ',' ',' ','.',' ',' ',' ',' ',' ','\47',' '}; //
        Array donde se almacena la latitud del gps.
90  char Array_Latitud_sms[25]={'L','a','t',':',' ',' ',' ','g',' ',' ',' ','.',' ',' ',' '
        ,' ',' ',' ',' ','m',' '};     // Array donde se almacena la altitud del gps para
        enviar el sms.
91  char Array_Longitud[14]={' ',' ','\337',' ',' ',' ','.',' ',' ',' ',' ',' ','\47',' '}; //
        Array donde se almacena la longitud del gps.
92  char Array_Longitud_sms[25]={'L','o','n',':',' ',' ',' ','g',' ',' ',' ','.',' ',' ',' '
        ,' ',' ',' ',' ','m',' '};     // Array donde se almacena la longitud del gps para
        enviar el sms.
93  char Array_Altitud[14]={' ',' ',' ',' ',' ',' ',' ',' ',' ',' '};     // Array donde se
        almacena la altitud del gps.
94  char Array_Hora[10]={' ',' ',':',' ',' ',':',' ',' '};           // Array donde se
        almacena la hora del gps.
95  char Array_HN[10]={' ',' ',':',' ',' ',':',' ',' '};             // Array donde se
        almacena la hora nueva del gps.
96
97  // Variables relacionadas con los números de teléfono.
98
99  char numero_Tlf_1[12];      // Array donde permanece el número 1 de teléfono.
100 char numero_Tlf_1_Aux[12]; // Array del primer número de teléfono al cual se le
        vuelcan las variables introducidas por el usuario.
101 char numero_Tlf_2[12];      // Array donde permanece el número 2 de teléfono.
102 char numero_Tlf_2_Aux[12]; // Array del segundo numero de teléfono al cual se le
        vuelcan las variables introducidas por el usuario.
103 char numero_Tlf_3[12];      // Array donde permanece el número 3 de teléfono.
```

```
104   char numero_Tlf_3_Aux[12]; // Array del tercer  número de teléfono al cual se le
          vuelcan las variables introducidas por el usuario.
105   char Numero_PIN[6];        // Array donde permanece el número PIN de la tarjeta
          SIM insertada en el módulo.
106   char Numero_PIN_Aux[6];    // Array del número Pin al cual se le vuelcan las
          variables que va introduciendo el usuario.
107   char numero_buf[25];       // Array que almacena el comando AT que establece el
          número de teléfono.
108   char pin_buf[20];          // Array que almacena el comando AT que establece el
          número PIN de la tarjeta SIM insertada en el módulo.
109   char introduce_Num[2];     // almacena el número en modo char.
110
111   char array_Minutos[3];     // Array que almacena los minutos.
112
113   // Variables que recogen un SMS recibido
114
115   int8_t answer;
116   int x;
117   char SMS[100];             // Almacena toda la cadena del SMS recibido.
118   char *puntSms;             // Creamos un puntero de tipo char.
119   char *arrayPuntSms[5];     // y un array de punteros de tipo char.
120   char telefonoSms[16];      // Almacena el número de teléfono del SMS recibido.
121   char comillas[4];          // Almacena las comillas de la segunda parte del array.
122   char fechaSms[12];         // Almacena la fecha del SMS recibido.
123   char horaSms[14];          // Almacena la hora del SMS recibido.
124   char smsRecibido[4];       // Almacena el SMS recibido.
125
126   // Variables de tiempo
127
128   int Horas;                 // Guarda el valor entero de las horas.
129   int Horas_Aux;             // Guarda el valor entero de las horas para el cambio de
          hora.
130 //  int Horas_Aux1;
131   int Minutos;               // Almacena el valor entero de los minutos.
132   int Minutos_Aux;           // Almacena el valor entero de los minutos para su
          comparación.
133   int Segundos;              // Almacena el valor entero de los segundos.
134   int Segundos_Aux;          // Almacena el valor entero de los segundos para su
          comparación.
135   unsigned int t_Envio;      // Almacena el tiempo de envío.
136   int direccion=0;           // Dirección inicial de la EEPROM donde guardaré datos.
137   int f;                     // Variable que se incrementa para guardar los teléfonos
          en memoria EEPROM.
```

14.2. Configuración inicial: función “setup”

```
1 void setup()
2 {
```

```
3   lcd.begin(20,4);
4   lcd.clear();
5   delay(20);
6  //  lcd.init();                    // Inicializo el LCD.
7  //  lcd.backlight();               // Brillo de la pantalla encendido.
8   gps_inicio_pines () ;
9   pinMode(encoderPinA, INPUT_PULLUP); // Nuevo método para habilitar las
        resistencias de pullup.
10  pinMode(encoderPinB, INPUT_PULLUP);
11  pinMode(clearButton, INPUT_PULLUP);
12  /* Poner las resistencias de pullup activas (Método antiguo)*/
13  // digitalWrite(encoderPinA, HIGH);
14  // digitalWrite(encoderPinB, HIGH);
15  // digitalWrite(clearButton, HIGH);
16  /* Pin del encoder en la interrupción 0 (pin 2)*/
17  /* La interrupción 0 se encargara de ejecutar la función encod_A cuando ve un
        cambio sobre el pin A.*/
18  attachInterrupt(0, encod_A, CHANGE);
19  /* Pin del encoder en la interrupción 1 (pin 3)*/
20  /* La interrupción 1 se encargara de ejecutar la función encod_B cuando ve un
        cambio sobre el pin B.*/
21  attachInterrupt(1, encod_B, CHANGE);
22  /* Pin del pulsador en la interrupción 4 (pin 7)*/
23  /* La interrupción 4 se encargara de ejecutar la función seleccionar cuando ve un
        cambio sobre el pin 7.*/
24  attachInterrupt(4, seleccionar, CHANGE);
25  //  Recuperamos las variables guardadas en la EEPROM
26  direccion=0;
27  incremento = EEPROM.read(direccion);
28  direccion=1;
29  t_Envio = EEPROM.read(direccion);
30  direccion=33;
31  numSel=EEPROM.read(direccion);
32  direccion=34;
33  modoTrabajo=EEPROM.read(direccion);
34  direccion=0;
35  while (direccion!=9)
36  {
37  numero_Tlf_1[direccion] = EEPROM.read(direccion+2);
38  numero_Tlf_2[direccion] = EEPROM.read(direccion+11);
39  numero_Tlf_3[direccion] = EEPROM.read(direccion+20);
40  direccion++;
41  }
42  direccion=0;
43  while (direccion!=4)
44  {
45  Numero_PIN[direccion] = EEPROM.read(direccion+29);
46  direccion++;
47  }
```

```
48    direccion=0;
49    Minutos_Aux=65;
50    Segundos_Aux=65;
51    comenzar_gps () ;
52   }
```

14.3. Función “loop”

```
1  void loop()
2  {
3    rotating = true;  // reestablecer el circuito antirrebote
4    if (lastReportedPos != encoderPos)
5    {
6    lastReportedPos = encoderPos;
7    Actualiza_pantalla();
8    }
9
10   if (lastPosicion != posicion)
11   {
12   lastPosicion  = posicion;
13   menu_Seleccion();
14   lcd.clear();
15   delayMicroseconds(20000);
16   Actualiza_pantalla();
17   }
18   leer_gps ();
19 }
```

14.4. Resto de funciones utilizadas

14.4.1. Función “Actualiza_Pantalla()”

En esta función se encuentra todo el código fuente relacionado con lo que se visualiza por el display LCD.

```
1  void Actualiza_pantalla()
2  {
3    if (menu==-2)
4    {
5    lcd.setCursor(18,encoderPos+1);
6    lcd.print("<-");
7    if (encoderPos==0)
8    {lcd.setCursor(18,2); lcd.print("  ");}
9    if (encoderPos==1)
10   {lcd.setCursor(18,1); lcd.print("  ");}
11
12   lcd.setCursor(0,0);
```

```
13    lcd.print("***  BIENVENIDO  ***");
14    lcd.setCursor(0,1);
15    lcd.print("Usar PIN nuevo    ");
16    lcd.setCursor(0,2);
17    lcd.print("Usar PIN antiguo  ");
18    lcd.setCursor(0,3);
19    lcd.print("                    ");
20    goto Fin_Actualiza_Pantalla;
21    }
22
23    if(menu==-1)
24    {
25    lcd.setCursor(0,0);
26    lcd.print("   Introducir PIN  ");
27    lcd.setCursor(19,0);
28    lcd.print(c,DEC);
29    lcd.setCursor(0,1);
30    lcd.print("          ");
31    lcd.print(Numero_PIN_Aux);
32    lcd.setCursor(0,2);
33    lcd.print("-> 0123456789B  <-");
34    lcd.setCursor(encoderPos+4,3);
35    lcd.print("|");
36    lcd.setCursor(encoderPos+3,3);
37    lcd.print(" ");
38    lcd.setCursor(encoderPos+5,3);
39    lcd.print(" ");
40    if(encoderPos==0)
41    {
42    lcd.setCursor(14,3);
43    lcd.print(" ");
44    }
45    goto Fin_Actualiza_Pantalla;
46    }
47
48    else if (menu==0)
49    {
50    lcd.setCursor(0,0);
51    lcd.print("  El numero PIN se  ");
52    lcd.setCursor(0,1);
53    lcd.print("   ha introducido   ");
54    lcd.setCursor(0,2);
55    lcd.print("    correctamente   ");
56    lcd.setCursor(0,3);
57    lcd.print("Presione para salir ");
58    goto Fin_Actualiza_Pantalla;
59    }
60
61    else if(menu==1)
```

```
62    {
63    lcd.setCursor(15,0);
64    lcd.print(encoderPos,DEC);
65    lcd.print("/3  ");
66
67    if (encoderPos==0 || encoderPos==1 || encoderPos==2 )
68    {
69    lcd.setCursor(18,encoderPos+1);
70    lcd.print("<-");
71    // OPCIONES DE SELECCION
72    if (encoderPos+1==2 ||encoderPos+1==3)
73    {lcd.setCursor(18,1); lcd.print("  ");}
74    if (encoderPos+1==1 ||encoderPos+1==3)
75    {lcd.setCursor(18,2); lcd.print("  ");}
76    if (encoderPos+1==1 ||encoderPos+1==2)
77    {lcd.setCursor(18,3); lcd.print("  ");}
78    lcd.setCursor(0,1);
79    lcd.print("Datos GPS         ");
80    lcd.setCursor(0,2);
81    lcd.print("Ajuste hora       ");
82    lcd.setCursor(0,3);
83    lcd.print("Ajustes de envio  ");
84    }
85    else if(encoderPos==3 )
86    {
87    lcd.setCursor(18,encoderPos-2);
88    lcd.print("<-");
89    lcd.setCursor(0,1);
90    lcd.print("Modo de trabajo   ");
91    lcd.setCursor(0,2);
92    lcd.print("                    ");
93    lcd.setCursor(0,3);
94    lcd.print("                    ");
95    }
96    goto Fin_Actualiza_Pantalla;
97    }
98
99    else if(menu==2)
100   {
101   lcd.setCursor(15,0);
102   lcd.print(encoderPos,DEC);
103   lcd.print("/4  ");
104
105   if (encoderPos==0 || encoderPos==1 || encoderPos==2 )
106   {
107 // OPCIONES DE SELECCIÓN
108   lcd.setCursor(18,encoderPos+1);
109   lcd.print("<-");
110   if (encoderPos+1==2 ||encoderPos+1==3)
```

```
111   {lcd.setCursor(18,1); lcd.print("  ");}
112   if (encoderPos+1==1 ||encoderPos+1==3)
113   {lcd.setCursor(18,2); lcd.print("  ");}
114   if (encoderPos+1==1 ||encoderPos+1==2)
115   {lcd.setCursor(18,3); lcd.print("  ");}
116
117    if (Estado==0)
118    {
119    lcd.setCursor(0,1);
120    lcd.print("Lon : Calc        ");
121    lcd.setCursor(0,2);
122    lcd.print("Lat : Calc        ");
123    lcd.setCursor(0,3);
124    lcd.print("Sat : Calc        ");
125    }
126
127    if (Estado==1)
128    {
129    lcd.setCursor(0,1);
130    lcd.print("Lon: ");
131    lcd.setCursor(17,1);
132    lcd.print(" ");
133    lcd.setCursor(0,2);
134    lcd.print("Lat: ");
135    lcd.setCursor(17,2);
136    lcd.print(" ");
137    lcd.setCursor(0,3);
138    lcd.print("Sat: ");
139    lcd.setCursor(6,3);
140    lcd.print("            ");
141    }
142    }
143    else if(encoderPos==3 || encoderPos==4)
144    {
145 // OPCIONES DE SELECCIÓN
146    lcd.setCursor(18,encoderPos-2);
147    lcd.print("<-");
148    if (encoderPos-2==1)
149    {lcd.setCursor(18,2); lcd.print("  ");}
150    if (encoderPos-2==2)
151    {lcd.setCursor(18,1); lcd.print("  ");}
152 //
153    lcd.setCursor(0,2);
154    lcd.print("Salir             ");
155    lcd.setCursor(0,3);
156    lcd.print("                    ");
157
158    if (Estado==0)
159    {
```

```
160     lcd.setCursor(0,1);
161     lcd.print("Alt : Calc        ");
162     }
163     if (Estado==1)
164     {
165     lcd.setCursor(0,1);
166     lcd.print("Alt: ");
167     lcd.setCursor(14,1);
168     lcd.print("    ");
169     }
170     }
171     goto Fin_Actualiza_Pantalla;
172     }
173
174   else if(menu==3)
175   {
176   lcd.setCursor(0,0);
177   lcd.print("** CAMBIO DE HORA **");
178   lcd.setCursor(0,1);
179   lcd.print("Hora Actual:");
180   lcd.setCursor(0,2);
181   lcd.print("Hora Nueva: ");
182   lcd.setCursor(8,3);
183   lcd.print("GMT");
184   if (encoderPos <0 && encoderPos >-10)
185   {
186   lcd.setCursor(11,3);
187   lcd.print(encoderPos,DEC);
188   lcd.print(" ");
189   }
190   else if (encoderPos ==0)
191   {
192   lcd.setCursor(11,3);
193   lcd.print(" ");
194   lcd.print(encoderPos,DEC);
195   lcd.setCursor(13,3);
196   lcd.print(" ");
197   }
198   else if (encoderPos >0 && encoderPos <10)
199   {
200   lcd.setCursor(11,3);
201   lcd.print("+");
202   lcd.print(encoderPos,DEC);
203   lcd.print(" ");
204   }
205   else if ( encoderPos >=10)
206   {
207   lcd.setCursor(11,3);
208   lcd.print("+");
```

```
209   lcd.print(encoderPos,DEC);
210   }
211   else
212   {
213   lcd.setCursor(11,3);
214   lcd.print(encoderPos,DEC);
215   }
216   incremento_Aux=encoderPos;
217
218   goto Fin_Actualiza_Pantalla;
219   }
220
221   else if(menu==4)
222   {
223   lcd.setCursor(0,1);
224   lcd.print("   Hora cambiada    ");
225   lcd.setCursor(0,2);
226   lcd.print("   Correctamente    ");
227   lcd.setCursor(0,3);
228   lcd.print("Presione para salir ");
229   goto Fin_Actualiza_Pantalla;
230   }
231   else if(menu==5)
232   {
233   lcd.setCursor(18,encoderPos);
234   lcd.print("<-");
235
236   if (encoderPos==1 ||encoderPos==2||encoderPos==3)
237   {lcd.setCursor(18,0); lcd.print("  ");}
238   if (encoderPos==0 ||encoderPos==2||encoderPos==3)
239   {lcd.setCursor(18,1); lcd.print("  ");}
240   if (encoderPos==0 ||encoderPos==1||encoderPos==3)
241   {lcd.setCursor(18,2); lcd.print("  ");}
242   if (encoderPos==0 ||encoderPos==1||encoderPos==2)
243   {lcd.setCursor(18,3); lcd.print("  ");}
244
245   lcd.setCursor(0,0);
246   lcd.print("Seleccionar num   ");
247   lcd.setCursor(0,1);
248   lcd.print("Editar numero     ");
249   lcd.setCursor(0,2);
250   lcd.print("Tiempo de envio   ");
251   lcd.setCursor(0,3);
252   lcd.print("Salir             ");
253   goto Fin_Actualiza_Pantalla;
254   }
255   else if ( menu==6 || menu==10)
256   {
257   lcd.setCursor(18,encoderPos);
```

```
258   lcd.print("<-");
259
260   if (encoderPos==1 ||encoderPos==2||encoderPos==3)
261   {lcd.setCursor(18,0); lcd.print("  ");}
262   if (encoderPos==0 ||encoderPos==2||encoderPos==3)
263   {lcd.setCursor(18,1); lcd.print("  ");}
264   if (encoderPos==0 ||encoderPos==1||encoderPos==3)
265   {lcd.setCursor(18,2); lcd.print("  ");}
266   if (encoderPos==0 ||encoderPos==1||encoderPos==2)
267   {lcd.setCursor(18,3); lcd.print("  ");}
268
269   lcd.setCursor(0,0);
270   lcd.print("Num 1:   ");
271   lcd.print(numero_Tlf_1);
272   lcd.setCursor(0,1);
273   lcd.print("Num 2:   ");
274   lcd.print(numero_Tlf_2);
275   lcd.setCursor(0,2);
276   lcd.print("Num 3:   ");
277   lcd.print(numero_Tlf_3);
278   lcd.setCursor(0,3);
279   lcd.print("Salir              ");
280   goto Fin_Actualiza_Pantalla;
281   }
282
283   else if(menu==7 || menu==8 || menu==9)
284   {
285   lcd.setCursor(0,0); // Posicionamos el cursor.
286   lcd.print("     El numero      ");
287   lcd.setCursor(5,1);
288   if (menu==7){lcd.print(numero_Tlf_1);}
289   if (menu==8){lcd.print(numero_Tlf_2);}
290   if (menu==9){lcd.print(numero_Tlf_3);}
291   lcd.setCursor(0,2);
292   lcd.print("Ha sido seleccionado");
293   lcd.setCursor(0,3);
294   lcd.print("Presione para salir ");
295   goto Fin_Actualiza_Pantalla;
296   }
297
298   else if(menu==11 || menu==12 || menu==13)
299   {
300   lcd.setCursor(0,0);
301   lcd.print("Actual:  ");
302   if (menu==11){lcd.print(numero_Tlf_1);}
303   if (menu==12){lcd.print(numero_Tlf_2);}
304   if (menu==13){lcd.print(numero_Tlf_3);}
305   lcd.setCursor(19,0);
306   lcd.print(c,DEC);
```

```
307   lcd.setCursor(0,1);
308   lcd.print("Nuevo:    ");
309   if (menu==11){lcd.print(numero_Tlf_1_Aux);}
310   if (menu==12){lcd.print(numero_Tlf_2_Aux);}
311   if (menu==13){lcd.print(numero_Tlf_3_Aux);}
312   lcd.setCursor(0,2);
313   lcd.print("->  0123456789B   <-");
314   lcd.setCursor(encoderPos+4,3);
315   lcd.print("|");
316   lcd.setCursor(encoderPos+3,3);
317   lcd.print(" ");
318   lcd.setCursor(encoderPos+5,3);
319   lcd.print(" ");
320   if(encoderPos==0)
321   {
322   lcd.setCursor(14,3);
323   lcd.print(" ");
324   }
325   goto Fin_Actualiza_Pantalla;
326   }
327
328   else if(menu==14 || menu==15 || menu==16)
329   {
330   lcd.setCursor(0,0);
331   lcd.print(" El numero ha sido  ");
332   lcd.setCursor(0,1);
333   lcd.print("      Cambiado      ");
334   lcd.setCursor(5,2);
335   if (menu==14){lcd.print(numero_Tlf_1_Aux);}
336   if (menu==15){lcd.print(numero_Tlf_2_Aux);}
337   if (menu==16){lcd.print(numero_Tlf_3_Aux);}
338   lcd.setCursor(0,3);
339   lcd.print("Presione para salir ");
340   goto Fin_Actualiza_Pantalla;
341   }
342
343   else if(menu==17)
344   {
345   lcd.setCursor(0,0);
346   lcd.print("* Tiempo de envio  *");
347   lcd.setCursor(0,1);
348   lcd.print(" Enviar SMS cada ");
349   lcd.print(encoderPos,DEC);
350   lcd.print(" ");
351   lcd.setCursor(0,2);
352   lcd.print("    min y 20 Seg    ");
353   lcd.setCursor(0,3);
354   lcd.print("Presione para salir ");
355   goto Fin_Actualiza_Pantalla;
```

```
356   }
357
358   else if(menu==18)
359   {
360    lcd.setCursor(18,encoderPos+1);
361   lcd.print("<-");
362   if (encoderPos+1==2 ||encoderPos+1==3)
363   {lcd.setCursor(18,1); lcd.print("  ");}
364   if (encoderPos+1==1 ||encoderPos+1==3)
365   {lcd.setCursor(18,2); lcd.print("  ");}
366   if (encoderPos+1==1 ||encoderPos+1==2)
367   {lcd.setCursor(18,3); lcd.print("  ");}
368
369   lcd.setCursor(0,1);
370   lcd.print("Modo autonomo     ");
371   lcd.setCursor(0,2);
372   lcd.print("Peticion usuario  ");
373   lcd.setCursor(0,3);
374   lcd.print("Brujula digital   ");
375   goto Fin_Actualiza_Pantalla;
376   }
377 //
378   else if (menu==19 || menu==20 || menu==21)
379   {
380   lcd.setCursor(0,0);
381   lcd.print("                  ");
382   lcd.setCursor(0,1);
383   if (menu==19){lcd.print("   Modo autonomo    ");}
384   if (menu==20){lcd.print("   Modo peticion    ");}
385   if (menu==21){lcd.print("Modo brujula digital");}
386   lcd.setCursor(0,2);
387   lcd.print("    seleccionado    ");
388   lcd.setCursor(0,3);
389   lcd.print("Presione para salir ");
390   goto Fin_Actualiza_Pantalla;
391   }
392 Fin_Actualiza_Pantalla:;
393 }
```

14.4.2. Función “menu_Seleccion()”

En esta función se realizan operaciones con las variables de control necesarias para la transición entre los menús.

```
1    void menu_Seleccion()
2 {
3    if (menu==-2 && encoderPos==0 && posicion!=0)
4    {menu=-1; encoderPos=0; goto Fin_seleccionar;}
5
```

```
6     if (menu==-2 && encoderPos ==1)
7     {menu=1; encoderPos=0; Flag_start=true; goto Fin_seleccionar;}
8
9     if(menu==-1 && posicion!=0)
10    {
11    if (encoderPos==10)
12    {
13    c=c-1;
14    if (c <= 0){c=0;}
15    Numero_PIN_Aux[c]=' ';
16    }
17    else
18    {
19    Numero_PIN_Aux[c]=Num_Sel (encoderPos);
20    c=c+1;
21    if (c==4){menu=0;}
22    }
23    goto Fin_seleccionar;
24    }
25
26    if (menu==0)
27    {
28    c=0;
29    encoderPos=0;
30    Flag_start=true;
31    strncpy(Numero_PIN, Numero_PIN_Aux, 4); // Copia 4 chars entre las posiciones
          0-3.
32    direccion=0 ;
33    while (direccion!=4)
34    {
35    EEPROM.write( direccion+29, Numero_PIN[direccion]);
36    direccion++;
37    }
38    direccion=0 ;
39    goto Fin_seleccionar;
40    }
41
42    if(menu==1 && encoderPos==0)
43    { menu=2; encoderPos=0; goto Fin_seleccionar;}
44
45    if (menu==1 && encoderPos==1)
46    { menu=3; encoderPos=0; goto Fin_seleccionar;}
47
48    if (menu==1 && encoderPos==2)
49    { menu=5; encoderPos=0; goto Fin_seleccionar;}
50
51    if (menu==1 && encoderPos==3)
52    { menu=18; encoderPos=modoTrabajo; goto Fin_seleccionar;}
53
```

```
54      if (menu==2 && encoderPos==4)
55      { menu=1; encoderPos=0; goto Fin_seleccionar;}
56
57      // Accede al menú de aceptar, cancelar
58      // o volver a cambiar la hora
59      if (menu==3)
60      { menu=4; encoderPos=0; goto Fin_seleccionar;}
61
62      // Guarda el cambio de hora en EEPROM
63      // y sale al menú principal.
64      if (menu==4 )
65      {
66      incremento=incremento_Aux;
67      EEPROM.write(direccion,incremento);
68      menu=1;
69      encoderPos=0;
70      goto Fin_seleccionar;
71      }
72
73      // Accede al menú de ajustes de envío
74      if (menu==5 && encoderPos==0)
75      {menu=6; encoderPos=0; goto Fin_seleccionar;}
76
77      // Accede a la edición de los números 1,2 y 3
78      if (menu==5 && encoderPos==1)
79      {menu=10; encoderPos=0; goto Fin_seleccionar;}
80
81      // Cambio del tiempo de envío
82       if (menu==5 && encoderPos==2)
83      {menu=17; encoderPos=t_Envio; goto Fin_seleccionar;}
84
85      // Opción de salir al menú principal
86      // desde el menú de ajustes de envió.
87      if (menu==5 && encoderPos==3)
88      {menu=1; encoderPos=0; goto Fin_seleccionar;}
89
90      // Menú de selección del número 1.
91      if (menu==6 && encoderPos==0)
92      {menu=7; encoderPos=0; goto Fin_seleccionar;}
93
94      // Menú de selección del número 2.
95      if (menu==6 && encoderPos==1)
96      {menu=8; encoderPos=0; goto Fin_seleccionar;}
97
98      // Menú de selección del número 3.
99      if (menu==6 && encoderPos==2)
100     {menu=9; encoderPos=0; goto Fin_seleccionar;}
101
102      // Regresa al menú ajustes de envío
```

```
103     if (menu==6 && encoderPos==3)
104     {menu=5; encoderPos=0; goto Fin_seleccionar;}
105
106     // Si presionamos el pulsador volveremos al menú 5.
107     // Numero 1 seleccionado.
108     if (menu==7)
109     {
110     menu=5;
111     numSel=1;
112     direccion=33;
113     EEPROM.write(direccion,numSel);
114     direccion=0;
115     encoderPos=0;
116     goto Fin_seleccionar;
117     }
118     // Si presionamos el pulsador volveremos al menú 5.
119     // Numero 2 seleccionado.
120     if (menu==8)
121     {
122     menu=5;
123     numSel=2;
124     direccion=33;
125     EEPROM.write(direccion,numSel);
126     direccion=0;
127     encoderPos=0;
128     goto Fin_seleccionar;
129     }
130
131     // Si presionamos el pulsador volveremos al menú 5.
132     // Numero 3 seleccionado.
133     if (menu==9)
134     {
135     menu=5;
136     numSel=3;
137     direccion=33;
138     EEPROM.write(direccion,numSel);
139     direccion=0;
140     encoderPos=0;
141     goto Fin_seleccionar;
142     }
143
144     // Cambio de número de teléfono 1.
145     if (menu==10 && encoderPos==0)
146     {menu=11; encoderPos=0; goto Fin_seleccionar;}
147
148    // Cambio de número de teléfono 2.
149     if (menu==10 && encoderPos==1)
150     { menu=12; encoderPos=0; goto Fin_seleccionar;}
151
```

```
152   // Cambio de número de teléfono 3.
153    if (menu==10 && encoderPos==2)
154    { menu=13; encoderPos=0; goto Fin_seleccionar;}
155
156    // Regresa al menú ajustes de envío
157    if (menu==10 && encoderPos==3)
158    {menu=5; encoderPos=0; goto Fin_seleccionar;}
159
160    // menú de cambio de número.
161    if (menu==11 || menu==12 || menu==13)
162    {
163    //Borra el número anterior
164    if (encoderPos==10)
165    {
166    c=c-1;
167    if (c <= 0){c=0;}
168    if (menu==11) {numero_Tlf_1_Aux[c]=' ';}
169    if (menu==12) {numero_Tlf_2_Aux[c]=' ';}
170    if (menu==13) {numero_Tlf_3_Aux[c]=' ';}
171    }
172    else
173    {
174    if (menu==11) {numero_Tlf_1_Aux[c]=Num_Sel (encoderPos); }
175    if (menu==12) {numero_Tlf_2_Aux[c]=Num_Sel (encoderPos); }
176    if (menu==13) {numero_Tlf_3_Aux[c]=Num_Sel (encoderPos); }
177    c=c+1;
178    if (c==9 && menu==11){menu=14;}
179    if (c==9 && menu==12){menu=15;}
180    if (c==9 && menu==13){menu=16;}
181    }
182    goto Fin_seleccionar;
183    }
184
185    if (menu==14)
186    {
187    c=0;
188    menu=5;
189    encoderPos=0;
190    strncpy(numero_Tlf_1, numero_Tlf_1_Aux, 9); // Copia 9 chars entre las posiciones
           0-8.
191    direccion=0;
192    while (direccion!=9)
193    {
194    EEPROM.write( direccion+2, numero_Tlf_1[direccion]);
195    direccion++;
196    }
197    direccion=0 ;
198    goto Fin_seleccionar;
199    }
```

```
200
201     if (menu==15)
202     {
203     c=0;
204     menu=5;
205     encoderPos=0;
206     strncpy(numero_Tlf_2, numero_Tlf_2_Aux, 9); // Copia 9 chars entre las posiciones
            0-8.
207     direccion=0 ;
208     while (direccion!=9)
209     {
210     EEPROM.write( direccion+11, numero_Tlf_2[direccion]);
211     direccion++;
212     }
213     direccion=0 ;
214     goto Fin_seleccionar;
215     }
216
217     if (menu==16)
218     {
219     c=0;
220     menu=5;
221     encoderPos=0;
222     strncpy(numero_Tlf_3, numero_Tlf_3_Aux,9); // Copia 9 chars entre las posiciones
            0-8.
223     direccion=0 ;
224     while (direccion!=9)
225     {
226     EEPROM.write( direccion+20, numero_Tlf_3[direccion]);
227     direccion++;
228     }
229     direccion=0 ;
230     goto Fin_seleccionar;
231     }
232     // Guardo el tiempo de envío en la EEPROM
233     if (menu==17 )
234     {
235      t_Envio=encoderPos;
236      direccion=1;
237      EEPROM.write(direccion,t_Envio);
238      direccion=0;
239      menu=5;
240      encoderPos=0;
241      goto Fin_seleccionar;
242     }
243
244     if (menu==18 && encoderPos==0)
245     {menu=19; encoderPos=0; goto Fin_seleccionar;}
246
```

```
247     if (menu==18 && encoderPos==1)
248     {menu=20; encoderPos=0; goto Fin_seleccionar;}
249
250     if (menu==18 && encoderPos==2)
251     {menu=21; encoderPos=0; goto Fin_seleccionar;}
252
253     if (menu==19)
254     {
255     menu=1;
256     modoTrabajo=0;
257     direccion=34;
258     EEPROM.write(direccion,modoTrabajo);
259     direccion=0;
260     encoderPos=0;
261     goto Fin_seleccionar;
262     }
263
264     if (menu==20)
265     {
266     menu=1;
267     modoTrabajo=1;
268     direccion=34;
269     EEPROM.write(direccion,modoTrabajo);
270     direccion=0;
271     encoderPos=0;
272     goto Fin_seleccionar;
273     }
274
275     if (menu==21)
276     {
277     menu=1;
278     modoTrabajo=2;
279     direccion=34;
280     EEPROM.write(direccion,modoTrabajo);
281     direccion=0;
282     encoderPos=0;
283     goto Fin_seleccionar;
284     }
285
286 Fin_seleccionar:;
287     }
```

14.4.3. Función “leerSmsSim()”

En esta función se lee un SMS almacenado en la primera posición de memoria de nuestra tarjeta SIM. Si el SMS recibido es “OP”, y además el dispositivo localizador ha encontrado la ubicación, este enviará un SMS al número de teléfono seleccionado por el valor de una variable con el contenido de la posición (latitud, longitud y hora de localización) y borrará los

SMSs almacenados hasta ese momento.

```
1  void leerSmsSim()
2 {
3  //   Serial.println("Comenzando...");
4     EnviaComandoAT("AT", "OK", 1000);
5     EnviaComandoAT("AT+CPIN=5856", "OK", 2000);
6     EnviaComandoAT("AT+COPS?", "+COPS:", 2000);
7 //    Serial.println("Modo SMS...");
8     EnviaComandoAT("AT+CMGF=1", "OK", 1000);      // Sms modo texto
9     EnviaComandoAT("AT+CPMS=\"SM\"", "OK", 1000);   // Selecciona la memoria
10    answer = EnviaComandoAT("AT+CMGR=1", "+CMGR:", 2000);   // Lee el sexto Sms.
11    if (answer == 1)
12    {
13    answer = 0;
14 //    while(Serial.available() == 0);
15    // this loop reads the data of the SMS
16    do{
17    // Si hay datos en el buffer de entrada del UART, los leemos y los chequeamos
          para la respuesta.
18    if(Serial.available() > 0)
19    {
20    SMS[x] = Serial.read();
21    x++;
22
23    // Comprueba si la respuesta deseada (OK) es la respondida por el módulo.
24    if (strstr(SMS, "OK") != NULL){answer = 1;}
25    }
26    }while(answer == 0); // Esperamos por la respuesta del módulo con el time out.
27    SMS[x] = '\0';
28    Serial.print(SMS);
29    i=0;
30    x=0;
31    do
32    {
33    if (SMS[i]=='\r'){SMS[i]=',';}
34    if (SMS[i]=='\n'){SMS[i]=' ';}
35    i++;
36    }while(i<=101);
37    descomponerSms();
38    i=0;
39    x=0;
40    if (strcmp(smsRecibido," OP")==0 && Estado==1 && sat >=3)
41    {
42    EnviaComandoAT("AT+CMGD=1,4","OK",1000);
43
44    if (numSel==1){establecer_numero_tlf (numero_Tlf_1,Numero_PIN);} //
45    if (numSel==2){establecer_numero_tlf (numero_Tlf_2,Numero_PIN);} //
46    if (numSel==3){establecer_numero_tlf (numero_Tlf_3,Numero_PIN);} //
47    envia_mensaje_gsm (Array_Hora);          // Enviamos el SMS.
```

```
48     envia_mensaje_gsm (Array_Longitud_sms); // Enviamos el SMS.
49     envia_mensaje_gsm (Array_Latitud_sms);  // Enviamos el SMS.
50     finaliza_envio_gsm ();
51     }
52     }
53     else
54     {
55     Serial.print("error ");
56     Serial.println(answer, DEC);
57     }
58 }
```

14.4.4. Función “descomponerSms()”

En esta función se extrae toda la información de un SMS recibido, hora de recepción, fecha, número de teléfono desde el cual ha sido enviado el SMS y el contenido del propio SMS.

```
1   void descomponerSms()
2   {
3    i=0;
4    puntSms = strtok (SMS,",");  // Tokenizamos (troceamos) la cadena que tenemos en
         el array TramaGPG por las comas
5                                 // y el primer intervalo lo guardamos en pch (puntero
                                      char)
6   //   if (strcmp(puntSms," \"REC UNREAD\"")==0)    // Si es la correcta, seguimos
       adelante
7   //   {
8    while (puntSms != NULL)        // Mientras el dato sea válido, lo aceptamos para
         llenar el array GGA
9    {
10   puntSms = strtok (NULL, ","); // Pasamos al siguiente intervalo cortado de la
         cadena
11   arrayPuntSms[i]=puntSms;                    // Guardamos el valor de puntSms en el
         array GGA
12   i++;                           // Incrementamos el contador/acumulador
13   }
14   strncpy(telefonoSms,arrayPuntSms[0] , 14); // Copia 14 chars entre las posiciones
         0-13.
15   strncpy(comillas, arrayPuntSms[1], 2);      // Copia 2 chars entre las posiciones
         0-1.
16   strncpy(fechaSms, arrayPuntSms[2], 9);      // Copia 9 chars entre las posiciones
         0-8.
17   strncpy(horaSms, arrayPuntSms[3], 12);      // Copia 12 chars entre las posiciones
         0-11.
18   strncpy(smsRecibido, arrayPuntSms[4], 3);  // Copia 3 chars entre las posiciones
         0-2.
19  //   }
20   }
```

14.4.5. Función “establecer_numero_tlf()”

En esta función se envían los comandos “AT” necesarios para la comunicación del módulo vía SMS con el terminal móvil deseado.

A esta función se le pasan dos variables, el número de teléfono al cual se le desea enviar el SMS y el pin de la tarjeta SIM que utiliza el módulo.

```
1  // Establecer el numero de teléfono
2  void establecer_numero_tlf (char *numero,char *pin)
3  {
4    sprintf (numero_buf, "AT+CMGS=\"%s\"", numero);
5    sprintf (pin_buf, "AT+CPIN=\%s", pin);
6  //   habilitar_gsm() ;//Habilita el modo GSM.
7  //   deshabilitar_gps();//Deshabilita el modo GPS.
8    EnviaComandoAT("AT", "OK", 1000);
9    EnviaComandoAT("AT", "OK", 1000);
10   EnviaComandoAT(pin_buf, "OK", 2000);
11   EnviaComandoAT("AT+COPS?", "+COPS:", 2000); // Verifica la conexión.
12   EnviaComandoAT("AT+CMGF=1", "OK", 1000);     // Sms modo texto
13   EnviaComandoAT(numero_buf, "OK", 1000);      // Comando de número de teléfono.
14 }
```

14.4.6. Función “envia_mensaje_gsm()”

Esta función envía por el puerto serie el SMS deseado, el SMS a enviar se le pasa como variable, en este caso un array de chars.

```
1  // Envía el mensaje al teléfono móvil.
2  void envia_mensaje_gsm (char *mensaje)
3  {
4    Serial.println (mensaje);
5  }
```

14.4.7. Función “finaliza_envio_gsm()”

Esta función envía el terminador del SMS escrito“0X1A” en hexadecimal,(“26” en decimal).Luego de haber enviado el terminador, recibiremos el SMS descrito en la función anterior en nuestro terminal móvil.

```
1  // Envía el terminador de SMS
2  void finaliza_envio_gsm()
3  {
4    delay (1000);
5    Serial.write (26);
6    delay (2000);
7  //   deshabilitar_gsm() ;// Deshabilita el modo GSM.
8  //   habilitar_gps();     // Habilita el modo GPS.
9  }
```

14.4.8. Función “EnviaComandoAT”

Tal como se observa en algún código de los anteriores, cada vez que queremos enviar un comando “AT” a la shield GPS/GPRS/GSM V3.0 lo hacemos a través de la función “EnviaComandoAT”. Esta función recibe los siguientes parámetros:

1. El comando que queremos enviar.
2. La respuesta esperada frente a ese comando.
3. Tiempo límite para obtener esa respuesta antes de dar por supuesto que existe un posible error en la comunicación con el módulo, problemas de cobertura, que el módulo se encuentre apagado, que no se detecte ninguna tarjeta SIM en el módulo, etc.).

Además, esta función devuelve una respuesta (0 o 1) en función del éxito que haya tenido en el envío del comando.

```
1  int8_t EnviaComandoAT(char* comandoAT, char* resp_esperada, unsigned int timeout)
2  {
3   uint8_t x=0,  answer=0;
4   char respuesta[100];
5   unsigned long previous;
6   memset(respuesta, '\0', 100);  // Inicializa el string.
7   delay(100);
8   while( Serial.available() > 0) Serial.read();    // Limpia el buffer de entrada.
9   Serial.println(comandoAT);    // Envia el comando AT.
10  x = 0;
11  previous = millis();
12  // Este bucle espera por la respuesta.
13  do{
14  // Si hay datos en el buffer de entrada del UART, los leemos y los chequeamos para
        la respuesta.
15  if(Serial.available() != 0){
16  respuesta[x] = Serial.read();
17  x++;
18  // Comprueba si la respuesta esperada es la respuesta del módulo.
19  if (strstr(respuesta, resp_esperada) != NULL)
20  {
21  answer = 1;
22  }
23  }
24  // Espera por la respuesta con el time out
25  }while((answer == 0) && ((millis() - previous) < timeout));
26  return answer;
27 }
```

14.4.9. Función “calculo_hora()”

En esta función se obtiene la hora que recibe el módulo GPS a través de los satélites.

Para obtener la hora de forma correcta, a esta función se le pasa una variable tipo “char” que a su vez es un puntero que apunta siempre a la misma dirección de memoria, en la cual se encuentra la hora.La hora se almacenará en un array denominado “Array_Hora”.

Además de obtener la hora, descomponemos la misma en componentes tipo“int” , de las horas, minutos y segundos, para poder operar con ellos (controlar tiempos en el programa) y hacer posible el cambio de hora.

```
1  void calculo_hora(char *hora)
2    {
3      char array_Horas[3];      // Array que almacena las horas.
4      array_Horas[0]=hora[0]; // Guardamos en la posición 0 del array las decenas de
           las horas.
5      array_Horas[1]=hora[1]; // Guardamos en la posición 1 del array las unidades de
           las horas.
6      Horas_Aux=atoi(array_Horas);
7      Horas = atoi(array_Horas);
8      Horas=Horas+incremento;
9
10     if (Horas>=24 && Horas <=36){Horas=Horas-24;}
11     if (Horas>=-11 && Horas <0){Horas=Horas+24;}
12
13     if (Horas==0 ){Array_Hora[0]='0';Array_Hora[1]='0';}
14     if (Horas==1 ){Array_Hora[0]='0';Array_Hora[1]='1';}
15     if (Horas==2 ){Array_Hora[0]='0';Array_Hora[1]='2';}
16     if (Horas==3 ){Array_Hora[0]='0';Array_Hora[1]='3';}
17     if (Horas==4 ){Array_Hora[0]='0';Array_Hora[1]='4';}
18     if (Horas==5 ){Array_Hora[0]='0';Array_Hora[1]='5';}
19     if (Horas==6 ){Array_Hora[0]='0';Array_Hora[1]='6';}
20     if (Horas==7 ){Array_Hora[0]='0';Array_Hora[1]='7';}
21     if (Horas==8 ){Array_Hora[0]='0';Array_Hora[1]='8';}
22     if (Horas==9 ){Array_Hora[0]='0';Array_Hora[1]='9';}
23     if (Horas==10 ){Array_Hora[0]='1';Array_Hora[1]='0';}
24     if (Horas==11 ){Array_Hora[0]='1';Array_Hora[1]='1';}
25     if (Horas==12 ){Array_Hora[0]='1';Array_Hora[1]='2';}
26     if (Horas==13 ){Array_Hora[0]='1';Array_Hora[1]='3';}
27     if (Horas==14 ){Array_Hora[0]='1';Array_Hora[1]='4';}
28     if (Horas==15 ){Array_Hora[0]='1';Array_Hora[1]='5';}
29     if (Horas==16 ){Array_Hora[0]='1';Array_Hora[1]='6';}
30     if (Horas==17 ){Array_Hora[0]='1';Array_Hora[1]='7';}
31     if (Horas==18 ){Array_Hora[0]='1';Array_Hora[1]='8';}
32     if (Horas==19 ){Array_Hora[0]='1';Array_Hora[1]='9';}
33     if (Horas==20 ){Array_Hora[0]='2';Array_Hora[1]='0';}
34     if (Horas==21 ){Array_Hora[0]='2';Array_Hora[1]='1';}
35     if (Horas==22 ){Array_Hora[0]='2';Array_Hora[1]='2';}
36     if (Horas==23 ){Array_Hora[0]='2';Array_Hora[1]='3';}
37
38     array_Minutos[0]=hora[2]; // Guardamos en la posición 0 del array las decenas de
           las horas.
39     array_Minutos[1]=hora[3]; // Guardamos en la posición 1 del array las unidades
```

```
        de las horas.
40      Minutos = atoi(array_Minutos);
41      ////////////////////////////////////////////////
42      Array_Hora[3] = hora[2];
43      Array_Hora[4] = hora[3];
44      ////////////////////////////////////////////////
45      char array_Segundos[3];     // Array que almacena los segundos.
46      array_Segundos[0]=hora[4]; // Guardamos en la posición 0 del array las decenas
            de las horas.
47      array_Segundos[1]=hora[5]; // Guardamos en la posición 1 del array las unidades
            de las horas.
48      Segundos = atoi(array_Segundos);
49      Array_Hora[6]= hora[4];
50      Array_Hora[7]= hora[5];
51      Array_Hora[8]= '\0'; // En la posición 8 escribimos el terminador.
52    }
```

14.4.10. Función “calculo_hn()”

En esta función se trata de guardar en un array denominado “Array_HN” (Array_Hora Nueva) la nueva hora que introduce el usuario, manteniendo invariable la hora original que guarda el “Array_Hora”.

Para obtener la hora de forma correcta, a esta función se le pasa una variable tipo “char” que a su vez es un puntero que apunta siempre a la misma dirección de memoria, en la cual se encuentra la hora, la única variación con respecto a la función anterior es que durante el cambio de hora hacemos que una variable “incremento_Aux”se incremente o decremente a medida que giramos la rueda del encoder.

En el momento de aceptar la hora cambiada, la variable “incremento_Aux” se igualará a la variable “incremento”.

```
1  void calculo_hn(char *hora)
2    {
3      Horas_Aux=Horas_Aux+incremento_Aux;
4
5      if (Horas_Aux>=24 && Horas_Aux<=36){Horas_Aux=Horas_Aux-24;}
6      if (Horas_Aux>=-11 && Horas_Aux<0){Horas_Aux=Horas_Aux+24;}
7
8      ////////////////////////////////////////////////
9      if (Horas_Aux==0 ){ Array_HN[0]='0'; Array_HN[1]='0';}
10     if (Horas_Aux==1 ){ Array_HN[0]='0'; Array_HN[1]='1';}
11     if (Horas_Aux==2 ){ Array_HN[0]='0'; Array_HN[1]='2';}
12     if (Horas_Aux==3 ){ Array_HN[0]='0'; Array_HN[1]='3';}
13     if (Horas_Aux==4 ){ Array_HN[0]='0'; Array_HN[1]='4';}
14     if (Horas_Aux==5 ){ Array_HN[0]='0'; Array_HN[1]='5';}
15     if (Horas_Aux==6 ){ Array_HN[0]='0'; Array_HN[1]='6';}
16     if (Horas_Aux==7 ){ Array_HN[0]='0'; Array_HN[1]='7';}
17     if (Horas_Aux==8 ){ Array_HN[0]='0'; Array_HN[1]='8';}
```

```
18      if (Horas_Aux==9 ){ Array_HN[0]='0'; Array_HN[1]='9';}
19      if (Horas_Aux==10 ){ Array_HN[0]='1'; Array_HN[1]='0';}
20      if (Horas_Aux==11 ){ Array_HN[0]='1'; Array_HN[1]='1';}
21      if (Horas_Aux==12 ){ Array_HN[0]='1'; Array_HN[1]='2';}
22      if (Horas_Aux==13 ){ Array_HN[0]='1'; Array_HN[1]='3';}
23      if (Horas_Aux==14 ){ Array_HN[0]='1'; Array_HN[1]='4';}
24      if (Horas_Aux==15 ){ Array_HN[0]='1'; Array_HN[1]='5';}
25      if (Horas_Aux==16 ){ Array_HN[0]='1'; Array_HN[1]='6';}
26      if (Horas_Aux==17 ){ Array_HN[0]='1'; Array_HN[1]='7';}
27      if (Horas_Aux==18 ){ Array_HN[0]='1'; Array_HN[1]='8';}
28      if (Horas_Aux==19 ){ Array_HN[0]='1'; Array_HN[1]='9';}
29      if (Horas_Aux==20 ){ Array_HN[0]='2'; Array_HN[1]='0';}
30      if (Horas_Aux==21 ){ Array_HN[0]='2'; Array_HN[1]='1';}
31      if (Horas_Aux==22 ){ Array_HN[0]='2'; Array_HN[1]='2';}
32      if (Horas_Aux==23 ){ Array_HN[0]='2'; Array_HN[1]='3';}
33      //////////////////////////////////////////////////
34      Array_HN[3] = hora[2];
35      Array_HN[4] = hora[3];
36      //////////////////////////////////////////////////
37      Array_HN[6]= hora[4];
38      Array_HN[7]= hora[5];
39      Array_HN[8]= '\0'; // En la posición 8 escribimos el terminador.
40    }
```

14.4.11. Función “calculo_latitud()”

En esta función se trata de guardar en un array denominado “Array_Latitud”, la latitud en la que el dispositivo se encuentra.

Para obtener la latitud de forma correcta, a esta función se le pasan dos variables tipo “char” que a su vez son punteros que apuntan siempre a la misma dirección de memoria, en la cual se encuentra la latitud y la dirección (norte y sur) respectivamente.

```
1   void calculo_latitud(char *Latitud, char *Brujula2)
2 {
3   Array_Latitud[0]=Array_Latitud_sms[5]=Latitud[0];
4   Array_Latitud[1]=Array_Latitud_sms[6]=Latitud[1];
5   Array_Latitud[3]=Array_Latitud_sms[8]=Latitud[2];
6   Array_Latitud[4]=Array_Latitud_sms[9]=Latitud[3];
7   Array_Latitud[6]=Array_Latitud_sms[11]=Latitud[5];
8   Array_Latitud[7]=Array_Latitud_sms[12]=Latitud[6];
9   Array_Latitud[8]=Array_Latitud_sms[13]=Latitud[7];
10  Array_Latitud[9]=Array_Latitud_sms[14]=Latitud[8];
11  Array_Latitud_sms[15]=Latitud[9];
12  Array_Latitud_sms[16]=Latitud[10];
13  Array_Latitud[11]=Array_Latitud_sms[18]=*Brujula2;
14 }
```

14.4.12. Función “calculo_longitud()”

En esta función se trata de guardar en un array denominado “Array_Longitud”, la longitud en la que el dispositivo se encuentra.

Para obtener la longitud de forma correcta, a esta función se le pasan dos variables tipo “char” que a su vez son punteros que apuntan siempre a la misma dirección de memoria, en la cual se encuentra la latitud y la dirección (este y oeste) respectivamente.

```
1  void calculo_longitud(char *Longitud, char *Brujula1)
2  {
3    Array_Longitud[0]=Array_Longitud_sms[5]=Longitud[1];
4    Array_Longitud[1]=Array_Longitud_sms[6]=Longitud[2];
5    Array_Longitud[3]=Array_Longitud_sms[8]=Longitud[3];
6    Array_Longitud[4]=Array_Longitud_sms[9]=Longitud[4];
7    Array_Longitud[6]=Array_Longitud_sms[11]=Longitud[6];
8    Array_Longitud[7]=Array_Longitud_sms[12]=Longitud[7];
9    Array_Longitud[8]=Array_Longitud_sms[13]=Longitud[8];
10   Array_Longitud[9]=Array_Longitud_sms[14]=Longitud[9];
11   Array_Longitud_sms[15]=Longitud[10];
12   Array_Longitud_sms[16]=Longitud[11];
13   Array_Longitud[11]=Array_Longitud_sms[18]=*Brujula1;
14 }
```

14.4.13. Función “calculo_altitud()”

En esta función se trata de guardar en un array denominado “Array_Altitud”, la altitud en la que el dispositivo se encuentra.

Para obtener la altitud de forma correcta, a esta función se le pasa una variable tipo “char” que a su vez es un puntero que apunta siempre a la misma dirección de memoria, en la cual se encuentra la altitud.

```
1  void calculo_altitud(char *Altura,char*unidades )
2  {
3    Array_Altitud[0]=Altura[0];
4    Array_Altitud[1]=Altura[1];
5    Array_Altitud[2]=Altura[2];
6    Array_Altitud[3]=Altura[3];
7    Array_Altitud[4]=Altura[4];
8    Array_Altitud[5]=Altura[5];
9    Array_Altitud[6]=Altura[6];
10   Array_Altitud[8]=*unidades;
11 }
```

14.4.14. Función “comenzar_gps ()”

Esta función es una de las más importantes, en ella se establecen las directrices para el comienzo de todo el programa, estas directrices son:

1. Encendido del módulo.
2. Introducir el Pin de la tarjeta SIM insertada en el módulo y almacenarlo en un array para su posterior utilización.
3. Enviar los comandos “AT” necesarios para el funcionamiento inicial de el dispositivo.
4. Introducir el contenido de la primera visualización del LCD.

```
1  void comenzar_gps ()
2  {
3   digitalWrite(5,HIGH);
4   delay(1500);
5   digitalWrite(5,LOW);
6   delay(1500);
7   Serial.begin(9600);
8   habilitar_gsm ();
9   deshabilitar_gps ();
10  delay(2000);
11  //////////////// INTRODUCIR EL PIN ////////////////////
12  Serial.println("Introduciendo PIN");
13  while (Flag_start==false)
14  {
15  rotating = true;  // restablecer el circuito antirrebote
16  if (lastReportedPos != encoderPos)
17  {
18  lastReportedPos = encoderPos;
19  Actualiza_pantalla();
20  }
21
22  if (lastPosicion != posicion)
23  {
24  lastPosicion  = posicion;
25  menu_Seleccion();
26  lcd.clear();
27  delayMicroseconds(20000);
28  Actualiza_pantalla();
29  }
30  }
31  ////////////////////////////////////////////////////
32  lcd.clear();  // Borramos la pantalla.
33  delay(20);
34  lcd.setCursor(0,1);
35  lcd.print("  ** Iniciando **   ");
36  Serial.println("Enviando comandos AT");
37  delay(2000); // GPS listo.
38  EnviaComandoAT("AT", "OK", 1000);
39  EnviaComandoAT("AT+CPIN=5856", "OK", 2000);
40  EnviaComandoAT("AT+COPS?", "+COPS:", 2000);
41  EnviaComandoAT("AT+CMGD=1,4", "OK", 1000);
```

```
42    EnviaComandoAT("AT", "OK", 1000);
43    //turn on GPS power supply
44    EnviaComandoAT("AT+CGPSPWR=1", "OK", 1000);
45    //Resetea el GPS  en modo autonomo.
46    EnviaComandoAT("AT+CGPSRST=1", "OK", 1000);
47    habilitar_gps();
48    deshabilitar_gsm();
49    delay(2000);
50    menu=1;
51    lcd.clear();
52    delay(20);
53    lcd.setCursor(15,0);
54    lcd.print("0/3  ");
55    lcd.setCursor(0,1);
56    lcd.print("Datos GPS         <-");
57    lcd.setCursor(0,2);
58    lcd.print("Ajuste hora          ");
59    lcd.setCursor(0,3);
60    lcd.print("Ajustes de envio     ");
61    Serial.println("Gps conectando");
62 }
```

14.4.15. Función "leer_gps ()"

En esta función leemos la información serie que el módulo recibe y la almacenamos en un array denominado "TramaGPG", una vez lleno dicho array lo mostramos en el monitor serie.

```
1  void leer_gps ()
2  {
3     if (Serial.available()>0)
4    {
5        byteGPS = Serial.read();
6        if (byteGPS != '$')      // Si  caracter leído es distinto a $ comenzanos a
             guardar en el array.
7        {
8        TramaGPG[i]=byteGPS;     // y lo almacenaremos en la cadena de caracteres en la
              posición que corresponda
9        i++;                     // e incrementamos en uno el contador.
10       }
11       if (byteGPS == '$')
12       {
13       TramaGPG[0]='$';  // En la posición 0 introducimos el símbolo del dólar para
             acotar la trama desde el principio.
14       Serial.println();
15       Serial.print(TramaGPG);
16       Serial.println();
17       cadena();
18       memset(TramaGPG, 0, sizeof(TramaGPG)); // Inicializa a cero la cadena para
             eliminar
```

```
19                                          // restos no deseados de lecturas
                                               anteriores
20        i=1; // Reseteamos la variable i;
21        }
22     }
23 }
```

14.4.16. Función “cadena()”

En esta función extraemos del array denominado como “TramaGPG” la trama GGA contenida en el mismo hacia un array de punteros denominado GGA, en esta trama (GGA), encontramos toda la información requerida para obtener la posición: latitud, Longitud, dirección, hora, número de satélites encontrados, altitud, si hemos encontrado la posición...

```
1  void cadena()
2    {
3     i=0;                                  // Inicializamos el contador
4     memset(GGA, 0, sizeof(GGA));          // Limpiamos el array GGA introduciendo en
          el todo ceros
5
6     pch = strtok (TramaGPG,",");          // Tokenizamos (troceamos) la cadena que
          tenemos en el array TramaGPG por las comas
7                                           // y el primer intervalo lo guardamos en
                                               pch (puntero char)
8
9     //Analizamos ese intervalo guardado en pch para ver si es la cadena que
          necesitamos
10    if (strcmp(pch,"$GPGGA")==0)          // Si es la correcta, seguimos adelante
11      {
12        while (pch != NULL)               // Mientras el dato sea válido, lo
              aceptamos para llenar el array GGA
13          {
14            pch = strtok (NULL, ",");     // Pasamos al siguiente intervalo cortado
                  de la cadena
15            GGA[i]=pch;                   // Guardamos el valor de pch en el array
                  GGA
16            i++;                          // Incrementamos el contador/acumulador
17          }
18          Calcula(GGA,"$GPGGA");          // llamamos a la función que nos va a
                mostrar los datos, bien por serial o por LCD
19                                          // a esta función se le pasan dos parámetros
                                               1º el array de chars, 2º la cadena a
                                               comparar.
20      }
21  }
```

14.4.17. Función “Calcula”

Es la función más importante de todo el programa, en ella se establece el control de todas las funciones realizadas en el programa, se obtienen las variables de localización, envía los SMSs según el modo de trabajo que se tenga configurado, se muestra la hora en una posición concreta del LCD dependiendo del menú en el que se encuentre, etc.

```
1  void Calcula(char **GGAPrint, char *trama)                    // Capturamos los datos
       pasados a la función
2 {
3   Estado = atoi(GGAPrint[5]);                                  // Transformamos el char que
       contiene el estado de la conexión a un entero.
4   sat = atoi(GGAPrint[6]);                                     // Transformamos el char que
       contiene el número de satélites a un entero.
5   calculo_latitud(GGAPrint[1],GGAPrint[2]);                    // Pasamos a Array_Latitud los
       datos que contienen la latitud.
6   calculo_longitud(GGAPrint[3],GGAPrint[4]);                   // Pasamos a Array_Longitud los
       datos que contienen la longitud.
7   calculo_altitud(GGAPrint[8],GGAPrint[9]);                    // Pasamos a Array_Altitud los
       datos que contienen la altitud.
8   calculo_hora(GGAPrint[0]);                                   // Pasamos a Array_Hora los datos
        que contienen la hora.
9   calculo_hn(GGAPrint[0]);
10
11  if (Segundos_Aux!=Segundos)
12  {
13    if (menu==1)
14    {lcd.setCursor(4,0);lcd.print(Array_Hora);}
15   ////////////////////////////////////////////////
16    if (menu==2 )
17    {
18    lcd.setCursor(4,0);
19    lcd.print(Array_Hora);
20    if (Estado==1 && (encoderPos==0 || encoderPos==1 || encoderPos==2))
21    {
22    lcd.setCursor(5,1);          // Posicionamos el cursor.
23    lcd.print(Array_Longitud); // Imprimimos la cadena.
24    lcd.setCursor(5,2);          // Posicionamos el cursor.
25    lcd.print(Array_Latitud);  // Imprimimos la cadena.
26    lcd.setCursor(5,3);          // Posicionamos el cursor.
27    lcd.print(sat);              // Imprimimos los satélites existentes por el lcd.
28    }
29    else if(Estado==1 && (encoderPos==3 || encoderPos==4))
30    {
31    lcd.setCursor(5,1);         // Posicionamos el cursor.
32    lcd.print(Array_Altitud); // Imprimimos la variable.
33    }
34    }
35 //    ////////////////////////////////////////////////
36    if (menu==3)
```

```
37      {
38       lcd.setCursor(12,1);
39       lcd.print(Array_Hora);
40       lcd.setCursor(12,2);
41       lcd.print(Array_HN);
42      }
43  //    ////////////////////////////////////////////////
44      if (menu==4)
45      {lcd.setCursor(6,0); lcd.print(Array_HN);}
46
47      if( menu==18 || menu==19 || menu==20 || menu==21)
48      {lcd.setCursor(6,0); lcd.print(Array_Hora);}
49
50      Segundos_Aux=Segundos;
51      }
52
53      if (Minutos_Aux!= Minutos )//Comprueba cada minuto
54      {
55      // Modo de trabajo autonomo
56      if (modoTrabajo==0 && (Minutos %t_Envio==0)&&  Estado==1 && sat >=3 )
57      {
58      habilitar_gsm ();
59      deshabilitar_gps ();
60      delayMicroseconds(500000);
61      if (numSel==1){establecer_numero_tlf (numero_Tlf_1,Numero_PIN);} //
62      if (numSel==2){establecer_numero_tlf (numero_Tlf_2,Numero_PIN);} //
63      if (numSel==3){establecer_numero_tlf (numero_Tlf_3,Numero_PIN);} //
64      envia_mensaje_gsm (Array_Hora);          // Enviamos el SMS.
65      envia_mensaje_gsm (Array_Longitud_sms); // Enviamos el SMS.
66      envia_mensaje_gsm (Array_Latitud_sms);  // Enviamos el SMS.
67      finaliza_envio_gsm ();
68      Minutos_Aux=Minutos;
69      deshabilitar_gsm ();
70      habilitar_gps ();
71      delayMicroseconds(500000);
72      }
73      // Modo peticion de usuario
74      if ( modoTrabajo==1)
75      {
76      habilitar_gsm ();
77      deshabilitar_gps ();
78      delayMicroseconds(500000);
79      leerSmsSim();
80      Minutos_Aux=Minutos;
81      deshabilitar_gsm ();
82      habilitar_gps ();
83      delayMicroseconds(500000);
84      }
85      }
```

```
86 }
```

14.4.18. Función “encod_A()”

En esta función gestionamos la interrupción externa 1 de nuestro Arduino (pin 3) conectado al pin denominado como A de nuestro encoder rotativo.

En este caso se determina que la dirección que toma nuestro encoder rotativo es hacia la derecha, por lo tanto la variable que hace seleccionar una opción u otra de nuestro menú (encoderPos), incrementa una posición.

Además de todo esto, se gestiona que los valores que se producen en esa variable (encoderPos) no sean mayores que ciertos valores que dependen del menú en el que nos encontremos, por ejemplo, en el menú cambio de hora, el huso horario máximo existente es +12, por lo tanto el valor de la variable en cuestión no debe ser nunca superior.

```
1  // Interrupción en A por cambio de estado.
2  void encod_A()
3  {
4    detachInterrupt(1);
5    detachInterrupt(0);
6    // Antirrebote
7    if ( rotating ) delay (1);  // esperar un poco hasta que se haga el rebote
8    // Prueba de transición, hubo cambio?
9    if( digitalRead(encoderPinA) != A_set ) // Anti-rebote, una vez más
10   {
11   A_set = !A_set;
12   if ( A_set && !B_set ) // Ajustar contador + si A conduce al B
13   {
14   encoderPos =encoderPos + 1;
15   if (menu!=3 && encoderPos<=0)
16   {
17   encoderPos=0;
18   }
19
20   if ( menu==-2 && encoderPos>=2)
21   {
22     encoderPos=0;
23   }
24   if (menu==1 && encoderPos>=4)
25   {
26   encoderPos=0;
27   }
28   if (menu==2 && encoderPos>=5)
29   {
30   encoderPos=0;
31   }
32   if (menu==3 && encoderPos>=13)
33   {
```

```
34    encoderPos=0;
35    }
36    if ((menu==5) && encoderPos>=4)
37    {
38    encoderPos=0;
39    }
40
41    if ((menu==6 || menu==10 )&& encoderPos>=4)
42    {
43    encoderPos=0;
44    }
45
46    if ((menu==-1||menu==11||menu==12||menu==13) && encoderPos>=11)
47    {
48    encoderPos=0;
49    }
50    if (menu==17 && encoderPos>=60)
51    {
52    encoderPos=1;
53    }
54
55    if (menu==18 && encoderPos>=3)
56    {
57    encoderPos=0;
58    }
59
60    }
61    rotating = false;  // no more debouncing until loop() hits again
62    }
63    attachInterrupt(1, encod_B, CHANGE);
64    attachInterrupt(0, encod_A, CHANGE);
65 }
```

14.4.19. Función “encod_B()”

En esta función gestionamos la interrupción externa 0 de nuestro Arduino (pin 2) conectado al pin denominado como A de nuestro encoder rotativo.

En este caso se determina que la dirección que toma nuestro encoder rotativo es hacia la izquierda, por lo tanto la variable que hace seleccionar una opción u otra de nuestro menú (encoderPos), decrementa una posición.

Además de todo esto, se gestiona que los valores que se producen en esa variable (encoderPos) no sean menores que ciertos valores que dependen del menú en el que nos encontremos, por ejemplo, en el menú cambio de hora, el huso horario mínimo existente es -11, por lo tanto el valor de la variable en cuestión no debe ser nunca inferior.

```
1 // Interrupción en B por cambio de estado, es idéntico al anterior.
2 void encod_B()
```

```
3 {
4   detachInterrupt(0);
5   detachInterrupt(1);
6   if ( rotating ) delay (1);
7   if( digitalRead(encoderPinB) != B_set )
8   {
9   B_set = !B_set;
10  // Ajustar contador + si A conduce al B
11  if( B_set && !A_set )
12  {
13  encoderPos = encoderPos - 1;
14  if (menu!=3 && encoderPos<=0)
15  {
16  encoderPos=0;
17  }
18  if ( menu==-2 && encoderPos>=2)
19  {
20    encoderPos=0;
21  }
22  if (menu==1 && encoderPos>=4)
23  {
24  encoderPos=0;
25  }
26  if (menu==2 && encoderPos>=5)
27  {
28  encoderPos=0;
29  }
30  if (menu==3 && encoderPos<=-12)
31  {
32  encoderPos=0;
33  }
34  if ((menu==5) && encoderPos>=4)
35  {
36  encoderPos=0;
37  }
38
39  if ((menu==6 || menu==10) && encoderPos>=4)
40  {
41  encoderPos=0;
42  }
43
44  if ((menu==-1||menu==11||menu==12||menu==13) && encoderPos>=11)
45  {
46  encoderPos=0;
47  }
48  if (menu==17 && encoderPos<=1)
49  {
50  encoderPos=1;
51  }
```

```
52
53    if (menu==18 && encoderPos>=3)
54    {
55    encoderPos=0;
56    }
57
58    }
59    rotating = false;
60    }
61    attachInterrupt(0, encod_A, CHANGE);
62    attachInterrupt(1, encod_B, CHANGE);
63  }
```

14.4.20. Función “seleccionar()”

En esta función gestionamos la interrupción externa 4 de nuestro Arduino (pin 19) conectado al interruptor que posee nuestro encoder rotativo (el interruptor de selección).

La gestión del menú se realiza mediante la variable “posición”, si se detecta un cambio en esta variable, se recurre a la función “menu_Seleccion ( )” que se encargara de visualizar por la pantalla del LCD el menú indicado.

```
1  void seleccionar()
2  {
3    if ( rotating ) {delay (1);}
4
5    if( digitalRead(clearButton )!= C_set  )
6    {
7    C_set = !C_set;
8    if( C_set  )
9    {
10   posicion += 1;
11   rotating = false;
12   }
13   }
14 }
```

14.4.21. Función “gps_inicio_pines ()”

En esta función se configuran los pines de control del módulo GPS como salidas.

```
1
2  // Establece los pines de trabajo del módulo GPS
3  /*
4   Si se desea cambiar los pines de control o
5   bien tenemos conflictos para conectar el módulo
6   en esos pines de Arduino , podremos extraer los
7   jumpers J10~J12 y unir los pines adyacentes al
8   borde de la placa a otros pines digitales.
```

```
 */
void gps_inicio_pines ()
{
    pinMode (6, OUTPUT);
    pinMode (4, OUTPUT);
    pinMode (5, OUTPUT);
}
```

14.4.22. Función "Num_Sel"

En esta función se hace una conversión de un numero entero comprendido de 0 a 9 que se le pasa como variable y nos devuelve la variable tipo char correspondiente.

```
char Num_Sel (int V09)
{
switch (V09)
{
  case 0:
  introduce_Num[0]='0';
  break;
  case 1:
  introduce_Num[0]='1';
  break;
  case 2:
  introduce_Num[0]='2';
  break;
  case 3:
  introduce_Num[0]='3';
  break;
  case 4:
  introduce_Num[0]='4';
  break;
  case 5:
  introduce_Num[0]='5';
  break;
  case 6:
  introduce_Num[0]='6';
  break;
  case 7:
  introduce_Num[0]='7';
  break;
  case 8:
  introduce_Num[0]='8';
  break;
  case 9:
  introduce_Num[0]='9';
  break;
}
return introduce_Num[0];
```

```
37 }
```

Capítulo 15

Código fuente del entorno Android

El presente capítulo no pretende enseñar a programar en el entorno Android, simplemente se muestra el código que hemos ejecutado en el compilador Android Studio para la elaboración de la aplicación.

La base de un programa para Android parte siempre de la misma estructura, estructura que explicaremos posteriormente.

El entorno de programación de Android, es un lenguaje de programación orientado a objetos, nada tiene que ver con otros lenguajes que hemos estudiado en la escuela como por ejemplo Visual Basic.

El entorno de Android es más complejo , sobre todo para las personas que no están acostumbradas a un lenguaje de tan alto nivel. Es cierto que en este tipo de lenguajes existen numerosas funciones ,activitys, fragments (pequeños programas realizados que funcionan dentro de uno principal, como por ejemplo el código de utilización de Google Maps, o bien proceder al envío de un SMS al pulsar un botón en la aplicación de nuestro terminal móvil) que nos facilitan mucho a la hora de programar.

Cabe comentar que en estos momentos el entorno de programación en Android está en auge, existe mucha gente que desarrolla aplicaciones para sí misma y para terceros.Los desarrolladores de estas aplicaciones tienen la posibilidad de exportar su proyecto a un archivo con extensión .apk que podrá ser instalado en los diferentes dispositivos móviles, tablets etc, esto no exime de que existan personas que desarrollen aplicaciones malignas.Con esto queremos decir que las aplicaciones que desarrollemos en Android , si queremos que puedan tener algún futuro, tendremos que subirlas tarde o temprano al Google Market, siempre bajo las condiciones de Google, ello incluye tambien los aspectos de seguridad.

Una de las cosas que tenemos que saber es que si utilizamos los servicios de formaciones como Google, que, en este caso así ha sido, tendremos que realizar un proceso tedioso y en el que surgen todo tipo de problemas y cuyas pautas para su realización se comentan a continuación:

1. Al realizar una aplicación , ya sea de dominio privado o publico, utilizando los servicios de Google, necesitamos una serie de permisos de los cuales es obligatorio disponer para

la aplicación funcione correctamente.

2. Hemos tenido que desarrollar la aplicación con el código de Google Maps comentado hasta el final del desarrollo de la aplicación.

3. Es necesario dotar a la aplicación a desarrollar de todos los permisos necesarios, como la utilización de Google Maps o el envío de SMSs.Todos los permisos necesarios y su sintaxis los podremos encontrar fácilmente en la web.

4. Una vez resuelta nuestra aplicación y con el código de Google Maps preparado para su compilación, tendremos que poseer una cuenta de Google en el caso de que no la tengamos, y acceder al siguiente enlace Web:

   `https://code.google.com/apis/console/`

   Entonces creamos un proyecto nuevo. El acceso web anterior es la pagina de"Google services ", en ella podremos activar los servicios de Google que necesitamos para nuestra aplicación, en este caso el de la API de google maps en su segunda versión.

   Con la ayuda de los siguientes enlaces conseguimos las claves que nos permitirán, compilar el proyecto (modo debug) y generar el archivo instalable para nuestro smartphone (modo release), además de lo anteriormente citado, estas claves hacen que la aplicación quede registrada en Google y de alguna forma pueden saber el autor que ha desarrollado dicha aplicación.

   a) Obtener la clave para el debug:
      - `http://www.aprendeandroid.com/l6/keyMaps.htm`

   b) Preparar la key para el apk.release (a partir de lo de exportar):
      - `http://javiergarbedo.es/linux/90-guadalinex/356-preparar-aplicacion-para-public`

   c) Obtener la release key para el maps API (se necesita la clave anterior del release):
      - `https://gstortoni.wordpress.com/2012/05/28/como-firmar-una-app-android-que-usa-`

   De no tener estas claves, el mapa de Google se mantendría invisible, estas claves además, nos sirven por si en un futuro deseamos subir nuestra aplicación a Android Market, aunque primeramente deberíamos hacernos desarrolladores de Google pagando previamente unas tasas, luego tendríamos que registrar el producto, en este caso la aplicación y en ultima instancia subirlo a Google Market.

5. El apartado anterior no es necesario si no utilizamos los servicios de de Google o si no deseamos subir nuestra aplicación a Android Market.

6. Aun con todos los pasos que hemos realizado anteriormente el mapa seguirá sin visualizarse, esto es debido a que es necesario importar una especie de "drivers" de OpenGL, cuyo código se describe al final de la sección del Manifest (15.1) y que representamos a continuación:

```
1 <uses-feature
2         android:glEsVersion="0x00020000"
3         android:required="true" />
```

Como hemos dicho con anterioridad, explicaremos brevemente las partes de las que consta un programa en Android sin profundizar demasiado, pero por lo menos para entender el concepto.

1. **Sección Manifest:** En esta sección es donde se aplican los permisos necesarios de los que debe disponer la aplicación en nuestro smartphone, así como las claves extraídas del registro de la aplicación en Google. Los permisos de los que debe disponer nuestra aplicación en nuestro smartphone son, entre otros,los siguientes:

   - Envío de SMSs.
   - Recepción de SMSs (lectura).
   - Posibilidad de utilización de Google Maps, con todas las características básicas que puede poseer, tales como hacer un zoom en el mapa, rotar el mapa, poner un marcador.
   - Utilización de internet.

   Cabe comentar que parte del código fuente en esta sección ya viene generado al crear el proyecto.

   En la sección 15.1 se presenta el código fuente correspondiente.

2. **Sección activity main:** En esta sección se suelen declarar los objetos que se van a presentar en el entorno gráfico de nuestra aplicación, y ordenar su posición y regular el tamaño para que los objetos queden centrados dentro de nuestra pantalla y no se superpongan entre ellos. Algunos de los objetos que se definen en esta sección a continuación son:

   - Los botones y el tamaño de letra existente dentro de los mismos.
   - El texto fijo que se mostrará por pantalla cuando pulsemos dichos botones así como el tipo de letra y su tamaño, lo que se denomina comúnmente en programación orientada objetos como una textbox.
   - El mapa y su tamaño, así como centrar su posición.

   En la sección 15.3 se presenta el código fuente correspondiente.

3. **Sección Main Activity:** En esta sección se presenta el cuerpo principal del programa, se importan a la misma los objetos utilizados y todas las opciones gráficas para que la aplicación pueda funcionar de forma correcta.

Como hemos dicho anteriormente, Android es un lenguaje de programación orientado a objetos, y, refiriéndonos a esta sección, cabe comentar que el añadir un botón, una Text box, o el mapa de Google, la sección de código que ocupa la parte de las importaciones, la añade el programa de forma automática, y podríamos obviarla.

También se declaran , se realizan y se utilizan funciones, como por ejemplo la de extraer de un SMS su contenido, convertir los grados, minutos y segundos de la posición enviada por el módulo, enviar SMSs, etc .

Contiene también el funcionamiento y el tratamiento de los botones que incluye la aplicación, es decir, lo que ocurrirá instantes después al pulsar un botón, tales como obtener y tratar el SMS obtenido, enviar un SMS, mostrar por pantalla los errores que se produzcan,por ejemplo al introducir un número de teléfono erróneo o incompleto.

En la sección 15.5 se presenta el código fuente correspondiente.Esta parte del código ha sido comentada para que pueda ser entendida con mayor facilidad.

4. **Sección Mapa layout:** Esta sección ya no es habitual encontrarla en un proyecto de los habituales, se ha añadido a mayores, esto es debido a que nuestra aplicación necesita contener un mapa para poder visualizar nuestra posición. El trozo de código que se encuentra en la sección 15.2 así como los permisos que necesita nuestro terminal móvil para poder ver el mapa de Google que se incluyen en la sección 15.1 no nos los hemos inventado, hemos buscado la información necesaria para añadir la segunda versión de la API de Google Maps en el siguiente enlace:

   `http://stackoverflow.com/questions/16978190/add-google-maps-api-v2-in-a-fragment`

5. **Sección del activity del mapa:** En esta sección, que tampoco es habitual encontrarla en proyectos habituales, en ella lo que hacemos es crear una clase de dominio público dentro del proyecto para que nos deje utilizar el fragmento de código descrito en el apartado anterior (sección 15.2) en cualquier parte del proyecto.

   En la sección 15.4 se presenta el código fuente correspondiente.

15.1. Código del manifest

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.usuario.gps_app" >
4
5      <application
6          android:allowBackup="true"
7          android:icon="@drawable/ic_launcher"
8          android:label="@string/app_name"
9          android:theme="@style/AppTheme" >
10         <activity
11             android:name=".MainActivity"
```

```
12              android:label="@string/app_name" >
13              <intent-filter>
14                  <action android:name="android.intent.action.MAIN" />
15
16                  <category android:name="android.intent.category.LAUNCHER" />
17              </intent-filter>
18          </activity>
19
20          <meta-data
21              android:name="com.google.android.gms.version"
22              android:value="@integer/google_play_services_version"/>
23          <meta-data
24              android:name="com.google.android.maps.v2.API_KEY"
25              android:value="AIzaSyCzFuNQCZq8Q4q9OWf1gYpBiJr8HmMNGUw"/>
26      </application>
27
28      <uses-permission android:name="android.permission.READ_SMS" />
29      <uses-permission android:name="android.permission.WRITE_SMS" />
30      <uses-permission android:name="android.permission.WRITE_SMS" />
31
32      <permission
33          android:name="com.example.usuario.gps_app.permission.MAPS_RECEIVE"
34          android:protectionLevel="signature"/>
35      <uses-permission
36          android:name="com.example.usuario.gps_app.permission.MAPS_RECEIVE"/>
37      <uses-permission
38          android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
39      <uses-permission android:name="android.permission.INTERNET"/>
40      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
41      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
42      <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
43      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
44
45      <uses-feature
46          android:glEsVersion="0x00020000"
47          android:required="true" />
48  </manifest>
```

15.2. Código del mapa layout

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
4      android:orientation="vertical" android:layout_width="match_parent"
5      android:layout_height="match_parent">
6      <FrameLayout
7          android:layout_width="match_parent"
8          android:layout_height="match_parent"
```

```
9           android:padding="2dp"
10          >
11          <com.google.android.gms.maps.MapView
12              android:id="@+id/mapview"
13              android:layout_width="match_parent"
14              android:layout_height="match_parent"
15              />
16      </FrameLayout>
17  </LinearLayout>
```

15.3. Código del Activity Main

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent" android:paddingLeft="@dimen/
           activity_horizontal_margin"
5      android:paddingRight="@dimen/activity_horizontal_margin"
6      android:paddingTop="@dimen/activity_vertical_margin"
7      android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".
           MainActivity">
8
9      <LinearLayout
10         android:orientation="vertical"
11         android:layout_width="fill_parent"
12         android:layout_height="fill_parent">
13
14         <LinearLayout
15             android:orientation="horizontal"
16             android:layout_width="fill_parent"
17             android:layout_height="59dp">
18
19             <EditText
20                 android:layout_width="wrap_content"
21                 android:layout_height="wrap_content"
22                 android:inputType="phone"
23                 android:ems="10"
24                 android:id="@+id/editText2"
25                 android:layout_weight="1" />
26
27             <Button
28                 android:layout_width="wrap_content"
29                 android:layout_height="wrap_content"
30                 android:text="@string/btn_pos"
31                 android:id="@+id/btn_posicion"
32                 android:onClick="onClickObtener" />
33
34         </LinearLayout>
```

```
35
36          <LinearLayout
37              android:orientation="horizontal"
38              android:layout_width="fill_parent"
39              android:layout_height="56dp">
40
41              <EditText
42                  android:layout_width="wrap_content"
43                  android:layout_height="wrap_content"
44                  android:inputType="phone"
45                  android:ems="10"
46                  android:id="@+id/editText"
47                  android:visibility="visible" />
48
49              <Button
50                  android:layout_width="wrap_content"
51                  android:layout_height="wrap_content"
52                  android:text="@string/btn_peticion"
53                  android:id="@+id/btn_peticion"
54                  android:onClick="onClickPedir"
55                  android:nestedScrollingEnabled="true" />
56
57          </LinearLayout>
58          <LinearLayout
59              android:layout_width="wrap_content"
60              android:layout_height="wrap_content">
61
62              <TextView
63                  android:layout_width="wrap_content"
64                  android:layout_height="wrap_content"
65                  android:text="@string/txt_label"
66                  android:id="@+id/LabelHora" />
67              </LinearLayout>
68          <LinearLayout
69              android:layout_width="wrap_content"
70              android:layout_height="wrap_content">
71              <fragment
72                  android:id="@+id/map"
73                  android:name="com.google.android.gms.maps.SupportMapFragment"
74                  android:layout_width="match_parent"
75                  android:layout_height="match_parent"
76                  android:tag="tag_fragment_map" />
77          </LinearLayout>
78
79      </LinearLayout>
80
81  </RelativeLayout>
```

15.4. Código del activity del mapa

```
1  package com.example.usuario.gps_app;
2
3  import com.google.android.gms.maps.MapFragment;
4
5  /**
6   * Created by usuario on 27/04/2015.
7   */
8  public class MapaActivity extends MapFragment {
9  }
```

15.5. Main Activity

```
1  package com.example.usuario.gps_app;
2
3  import android.content.ContentResolver;
4  import android.database.Cursor;
5  import android.graphics.Color;
6  import android.net.Uri;
7  import android.support.v7.app.ActionBarActivity;
8  import android.os.Bundle;
9  import android.telephony.SmsManager;
10 import android.view.Menu;
11 import android.view.MenuItem;
12 import android.view.View;
13 import android.widget.Button;
14 import android.widget.EditText;
15 import android.widget.TextView;
16 import android.widget.Toast;
17
18
19 import com.google.android.gms.maps.CameraUpdate;
20 import com.google.android.gms.maps.CameraUpdateFactory;
21 import com.google.android.gms.maps.GoogleMap;
22 import com.google.android.gms.maps.SupportMapFragment;
23 import com.google.android.gms.maps.model.CameraPosition;
24 import com.google.android.gms.maps.model.LatLng;
25 import com.google.android.gms.maps.model.MarkerOptions;
26
27
28 public class MainActivity extends ActionBarActivity {
29
30         Button botonObtener;
31         Button botonPedir;
32         TextView labelHora;
33         EditText textBoxObtener;
```

```
34          EditText textBoxTelefono;
35          String   num_predet="615934713"; //número predeterminado que aparecerá al
                iniciar
36          GoogleMap mapa;//mapa con el que se trabajará
37
38
39      @Override
40          protected void onCreate(Bundle savedInstanceState) { //lo que hace la
                aplicación al iniciar
41              super.onCreate(savedInstanceState);
42              setContentView(R.layout.activity_main); //carga la view en el activity
43
44              botonObtener=(Button)findViewById(R.id.btn_posicion); //establecemos las
                     referencias de los elementos de la view en una variable para
                    gestionar
45              botonPedir=(Button)findViewById(R.id.btn_peticion);
46              textBoxObtener=(EditText)findViewById(R.id.editText2);
47              labelHora=(TextView)findViewById(R.id.LabelHora);
48              textBoxTelefono=(EditText)findViewById(R.id.editText);
49
50              textBoxObtener.setText(num_predet);//se establece en la caja de texto un
                     número predeterminado
51              textBoxTelefono.setText(num_predet);
52
53              mapa = ((SupportMapFragment) getSupportFragmentManager().
                    findFragmentById(R.id.map)).getMap();//Se relaciona al fragment de
                    tipo SupportMapFragment del xml del main
54              mapa.setMapType(GoogleMap.MAP_TYPE_HYBRID);
55          }
56
57          @Override
58          public boolean onCreateOptionsMenu(Menu menu) {
59              // Inflate the menu; this adds items to the action bar if it is present.
60              getMenuInflater().inflate(R.menu.menu_main, menu);
61              return true;
62          }
63
64          @Override
65          public boolean onOptionsItemSelected(MenuItem item) {
66              // Handle action bar item clicks here. The action bar will
67              // automatically handle clicks on the Home/Up button, so long
68              // as you specify a parent activity in AndroidManifest.xml.
69              int id = item.getItemId();
70
71              //noinspection SimplifiableIfStatement
72              if (id == R.id.action_settings) {
73                  return true;
74              }
75
```

```
76              return super.onOptionsItemSelected(item);
77          }
78
79          /**
80           * Método que obtiene las coordenadas a partir del mensaje de
81           * la propia bandeja del teléfono.
82           * Se debe de introducir un número válido en el text box que lo acompaña
83           * */
84          public void onClickObtener(View view){
85
86              String mensaje=obtenerMensaje(textBoxObtener.getText().toString());//
                    guardamos el mensaje en una variable
87              if(textBoxObtener.getText().length()<9){//comprueba si lo que hay en la
                    caja de texto tiene menos caracteres que un número telefónico
88                  Toast.makeText(getApplicationContext(),
89                          "Debe introducir un número de teléfono",
90                          Toast.LENGTH_LONG).show();
91                  textBoxObtener.setBackgroundColor(Color.rgb(255,150,150));//indica
                        que está mal lo que ahí se escribió
92              }else {
93                  if (textBoxObtener.getText().length() >= 10) {//si el número
                        introducido es mayor que un número de teléfono
94                      Toast.makeText(getApplicationContext(),
95                              "El número debe de ser válido (menor de 9 dígitos)" +
                                    mensaje.length(),
96                              Toast.LENGTH_LONG).show();
97                      textBoxObtener.setBackgroundColor(Color.rgb(100, 0, 0));
98                  } else {//si el número de teléfono es correcto
99                      textBoxObtener.setBackgroundColor(getResources().getColor(
                            android.R.color.transparent));//quitamos el color erróneo
                            del campo de texto
100                     try {//inicia la captura de excepciones
101                         String hora = mensaje.substring(0, 9); //obtenemos la hora
                                del mensaje
102                         String lon = mensaje.substring((mensaje.lastIndexOf("Lon:")
                                + 4), (mensaje.lastIndexOf("Lon:") + 19));//+2 si
                                extendida
103                         String lat = mensaje.substring((mensaje.lastIndexOf("Lat:")
                                + 4), (mensaje.lastIndexOf("Lat:") + 18));//+2 //
                                Obtenemos latitud y longitud del mensaje
104                         String lon_orientacion = lon.substring(lon.length() - 1);//
                                Obtenemos el carácter que indica cual es la orientación
                                W/E-S/N
105                         String lat_orientacion = lat.substring(lat.length() - 1);
106
107                         lon=lon.replaceAll("[^0-9]", ""); //limpiamos las Strings
                                correspondiente a las coordenadas de cualquier carácter
                                que no sea un dígito
108                         lat=lat.replaceAll("[^0-9]", "");
```

```
109
110                         int lonGrados=Integer.parseInt(lon.substring(0,2)); //se
                                obtienen los grados, es decir, los dos primeros dígitos
                                de esa string de dígitos
111                         float lonMinutos=Float.parseFloat(lon.substring(2))/1000000;
                                //se obtienen los minutos, y como previamente al purgar
                                de caracteres no numéricos la String quitamos el punto
                                decimal, se lo incorporamos dividiendo
112                         int latGrados=Integer.parseInt(lat.substring(0,2));
113                         float latMinutos=Float.parseFloat(lat.substring(2))/1000000;
114
115                         double longitud=conversionGradosDecimal(lonGrados,lonMinutos
                                );//Obtenemos las coordenadas reales en decimal a través
                                 de un método de conversión
116                         double latitud=conversionGradosDecimal(latGrados,latMinutos)
                                ;
117
118                        if (lon_orientacion.equalsIgnoreCase("W")) {
119                             longitud = longitud * (-1);//adaptamos el símbolo de las
                                     coordenadas numéricas dependiendo de la orientación
120                         }
121                         if (lat_orientacion.equalsIgnoreCase("S")) {
122                             latitud = latitud * (-1);
123                         }
124
125                         obtenerPosicion(latitud, longitud, hora); //llamamos al
                                método de obtener posición
126
127                         this.labelHora.setText("Hora de la localización: " + hora +
                                "\n Coordenadas: Lat" + conversionDecimalGrados(latitud)
                                 + "Lon" + conversionDecimalGrados(longitud));//
                                actualizamos la label con la hora y las coordenadas  lat
                                 43.176224 lon-8.203902  diferencia de lat 0.117502 y
                                long 0.136005
128                 }catch(StringIndexOutOfBoundsException e){//Si el número no tiene
                        mensajes o el último mensaje guardado no es del formato
                        establecido, captura el error.
129                         Toast.makeText(getApplicationContext(),
130                                 "Ese número no tiene un mensaje de coordenadas",
131                                 Toast.LENGTH_LONG).show();
132                     }
133                     }
134                 }
135         }
136
137     /**
138      * Método que calcula a partir de grados y minutos su valor decimal para
139      * Google Maps
140      * */
```

```
141     public double conversionGradosDecimal( double grados, double minutos )
142     {
143         return grados + (minutos/60);
144     }
145
146     public String conversionDecimalGrados(double coordenadas){
147         int segundos = (int)Math.round(coordenadas * 3600);
148         int grados = segundos / 3600;
149         segundos = Math.abs(segundos %3600);
150         int minutos = segundos / 60;
151         segundos %= 60;
152
153         return grados+"º "+minutos+"' "+segundos+"\"";
154     }
155
156         /**
157          * Enviamos un mensaje para que el dispositivo GPS nos envíe un mensaje con
158          * coordenadas. Escribimos el número al que queremos enviar
159          * */
160         public void onClickPedir(View view){
161             if(textBoxTelefono.getText().length()<9){//como en el anterior
162                 Toast.makeText(getApplicationContext(),
163                         "Debe introducir un número de teléfono",
164                         Toast.LENGTH_LONG).show();
165                 textBoxTelefono.setBackgroundColor(Color.rgb(100,0,0));
166             }else{
167                 if(textBoxTelefono.getText().length()>=10){
168                     Toast.makeText(getApplicationContext(),
169                             "El número debe de ser válido (menor de 9 dígitos)",
170                             Toast.LENGTH_LONG).show();
171                     textBoxTelefono.setBackgroundColor(Color.rgb(100,0,0));
172                 }else {
173
174                     sendSMS(textBoxTelefono.getText().toString());//llamamos al
                            método para enviar mensajes y le pasamos el número
175
176                     textBoxObtener.setText(textBoxTelefono.getText().toString());//
                            actualizamos las cajas para que tengan el mismo número
177                     textBoxTelefono.setBackgroundColor(getResources().getColor(
                            android.R.color.transparent));
178                 }
179             }
180         }
181
182         /**
183          * Envía un mensaje a un número que pasamos como parámetro
184          * */
185         protected void sendSMS(String numero) {
```

```
186             String toPhoneNumber = "+34"+numero; //número adaptado y ubicado en
                    españa
187             String smsMessage ="OP"; //mensaje a enviar
188             try {
189                 SmsManager smsManager = SmsManager.getDefault(); //crea un objeto
                        SmsManager con una instancia predeterminada
190                 smsManager.sendTextMessage(toPhoneNumber, null, smsMessage, null,
                        null); //se envía un mensaje a un numero dado
191                 Toast.makeText(getApplicationContext(), "SMS enviado.",
192                         Toast.LENGTH_LONG).show();
193             } catch (Exception e) {//si captura alguna excepción el mensaje no se
                    envía
194                 Toast.makeText(getApplicationContext(),
195                         "Envío de SMS fallido.",
196                         Toast.LENGTH_LONG).show();
197                 e.printStackTrace();
198             }
199         }
200
201         /**
202          * Obtiene el último mensaje en la bandeja de entrada del móvil
203          * para un número que le pasemos como parámetro
204          * */
205         protected String obtenerMensaje(String numero){
206             ContentResolver contentResolver = getContentResolver(); //Instanciamos un
                        objeto que gestiona los contenidos asociados al contexto de la
                        aplicación
207             String[] phoneNumber=new String[]{"+34"+numero}; //El número de teléfono
                    que queremos (array de string porque pueden ser varios)
208             Cursor cursor1 = contentResolver.query(Uri.parse("content://sms/inbox"),
                         new String[] { "_id", "thread_id", "address", "person", "date","
                        body", "type" }, "address=?", phoneNumber, null);
209             //Un cursor que recoge cada uno de los mensajes que el content resolver
                    nos pilla de la carpeta dada para el número dado
210
211             int indexBody = cursor1.getColumnIndex("body");
212             //En el cursor los valores se almacenan tal que asi: {([identificador]=
                    valor),([identficador2],valor2)}.
213             //Esto lo que hace es pillar el valor asociado a la columna que se llama
                         "body" donde irá el cuerpo del mensaje
214             if (indexBody < 0 || !cursor1.moveToFirst())//si no hay nada
215             {
216                 return "Nada";
217             }
218             else{
219                 return cursor1.getString(indexBody);//Escoge el primer body que
                        encuentre. El del primer mensaje en la bandeja de entrada, es
                        decir el último por fecha
220
```

```
221             }
222         }
223
224     /**
225      * Método que a partir de dos valores equivalentes a una medida de longitud y
             latitud
226      * situa el mapa en una posición y añade un marcador
227      * */
228     public void obtenerPosicion(double latitud, double longitud, String hora){
229
230         LatLng posicion = new LatLng(latitud, longitud);//Objeto de Google Maps que
                equivale a la latitud y longitud (+-gg.mmssss)
231         CameraPosition posicionCamara = new CameraPosition.Builder()//Objeto de
                cámara
232                 .target(posicion)    //Centramos el mapa en la posición
233                 .zoom(19)           //Establecemos el zoom en 19
234                 .bearing(45)        //Establecemos la orientación con el noreste
                        arriba
235                 .tilt(10)           //Bajamos el punto de vista de la cámara 10 grados
236                 .build();
237
238         CameraUpdate camara =
239                 CameraUpdateFactory.newCameraPosition(posicionCamara);//actualizamos
                        la cámara para que haga el movimiento
240         mapa.addMarker(new MarkerOptions()//añadimos un nuevo marcador al mapa en la
                posición que le dimos.
241                 .position(posicion)
242                 .title("posicion")
243                 .snippet(hora));
244         mapa.animateCamera(camara);//animamos la cámara con todos los datos
                anteriores
245     }
246 }
```

Capítulo 16

Compatibilidades del programa fuente con placas Arduino

16.1. Compatibilidad del programa fuente con Arduino Leonardo

A continuación comentaremos las secciones del programa que se muestra en el capítulo 14 en las que es necesario realizar unas modificaciones para poder hacer que funcione en Arduino Leonardo.

1. **Sección de inicialización de variables**

   Al principio del programa existe una sección en la cual se le asigna un nombre a los pines que posteriormente utilizaremos en el programa.

   La asignación de un nombre a cada pin que se utiliza dentro del programa se hace por comodidad, para no tener que recurrir al numero del pin utilizado cada vez que se requiere hacer una acción en función del estado del mismo.

   Uno de los pines que utiliza Arduino MEGA cambia con respecto a Arduino Leonardo, concretamente el botón de seleccionar, asociado a la interrupción externa 4 de Arduino MEGA (pin 19), en Leonardo, la interrupción externa 4 la implementa el pin 7, por tanto la sección de código a modificar es la siguiente:

```
1 enum PinAssignments
2   {
3   encoderPinA = 2,    // Derecha (Llamado DT en nuestro encoder)
4   encoderPinB = 3,    // Izquierda (Llamado CLK en nuestro encoder)
5   clearButton = 7,  // switch (Llamado SW en nuestro encoder)
6   /* Conectar +5v y Gnd en los pines restantes.*/
7   };
```

2. **Función "comenzar_gps ()"**

   En esta función es necesario incicializar la comunicación Serial1 de la cual dispone Arduino Leonardo y que utilizará para comunicarse con el módulo GPS.

Esto es debido a que el micro-controlador que utiliza Arduino Leonardo (32u4) usa Serial1 para comunicar con pines 0 y 1(hardware) el Serial está reservado para el puerto de serie virtual.

Debido al motivo anteriormente citado ,en el resto de funciones el principal cambio a realizar es el cambio en la comunicación, es decir cambiar Serial por Serial1.

La sección de código a añadir es la siguiente:

```
1 Serial1.begin(9600);
```

3. **Función "leer_gps ()"**

Como ya hemos dicho anteriormente, en Arduino Leonardo utiliza Serial1 para comunicarse, por tanto en esta sección es necesario cambiar la comprobación de si existen datos en el buffer de entrada y la posterior lectura de esos datos.

```
1  void leer\_gps ()
2  {
3     if (Serial1.available()>0)
4    {
5        byteGPS = Serial1.read();
6        if (byteGPS != '$')      // Si caracter leido es distinto a $ comenzanos
             a guardar en el array.
7        {
8        TramaGPG[i]=byteGPS;     // y lo almacenaremos en la cadena de caracteres
             en la posicion que corresponda
9        i++;                     // e incrementamos en uno el contador.
10       }
11       if (byteGPS == '$')
12       {
13       TramaGPG[0]='$';  // En la posicion 0 introducimos el simbolo del dolar
             para acotar la trama desde el principio.
14       Serial.println();
15       Serial.print(TramaGPG);
16       Serial.println();
17       cadena();
18       memset(TramaGPG, 0, sizeof(TramaGPG)); // Inicializa a cero la cadena
             para eliminar
19                                              // restos no deseados de lecturas
                                                    anteriores
20       i=1; // Reseteamos la variable i;
21       }
22    }
23 }
```

4. **Función "EnviaComandoAT"**

A continuación se adjunta el código modificado para el correcto funcionamiento de la función en Arduino Leonardo.

```
1  int8_t EnviaComandoAT(char* comandoAT, char* resp_esperada, unsigned int
       timeout)
2  {
3    uint8_t x=0,  answer=0;
4    char respuesta[100];
5    unsigned long previous;
6    memset(respuesta, '\0', 100);  // Inicializa el string.
7    delay(100);
8    while( Serial1.available() > 0) Serial1.read();    // Limpia el buffer de
         entrada.
9    Serial1.println(comandoAT);    // Envia el comando AT.
10   x = 0;
11   previous = millis();
12   // Este bucle espera por la respuesta.
13   do{
14   // Si hay datos en el buffer de entrada del UART, los leemos y los chequeamos
         para la respuesta.
15   if(Serial1.available() != 0){
16   respuesta[x] = Serial1.read();
17   x++;
18   // Comprueba si la respuesta esperada es la respuesta del módulo.
19   if (strstr(respuesta, resp_esperada) != NULL)
20   {
21   answer = 1;
22   }
23   }
24   // Espera por la respuesta con el time out
25   }while((answer == 0) && ((millis() - previous) < timeout));
26   return answer;
27  }
```

5. **Función “envia_mensaje_gsm”**

   A continuación se adjunta el código modificado para el correcto funcionamiento de la función en Arduino Leonardo.

```
1  // Envía el mensaje al teléfono móvil.
2  void envia_mensaje_gsm (char *mensaje)
3  {
4    Serial1.println (mensaje);
5  }
```

6. **Función “finaliza_envio_gsm()”**

   A continuación se adjunta el código modificado para el correcto funcionamiento de la función en Arduino Leonardo.

```
1  // Envia el terminador de SMS
2  void finaliza_envio_gsm()
```

```
3 {
4   delay (1000);
5   Serial1.write (26);
6   delay (2000);
7 //  deshabilitar_gsm() ;// Deshabilita el modo GSM.
8 //  habilitar_gps();     // Habilita el modo GPS.
9 }
```

7. **Función "leerSmsSim()"**

A continuación se adjunta el código modificado para el correcto funcionamiento de la función en Arduino Leonardo.

```
1  void leerSmsSim()
2 {
3  //  Serial.println("Comenzando...");
4    EnviaComandoAT("AT", "OK", 1000);
5    EnviaComandoAT("AT+CPIN=5856", "OK", 2000);
6    EnviaComandoAT("AT+COPS?", "+COPS:", 2000);
7 //   Serial.println("Modo SMS...");
8    EnviaComandoAT("AT+CMGF=1", "OK", 1000);     // Sms modo texto
9    EnviaComandoAT("AT+CPMS=\"SM\"", "OK", 1000);  // Selecciona la memoria
10   answer = EnviaComandoAT("AT+CMGR=1", "+CMGR:", 2000);  // Lee el sexto sms.
11   if (answer == 1)
12   {
13   answer = 0;
14 //   while(Serial.available() == 0);
15   // this loop reads the data of the SMS
16   do{
17   // Si hay datos en el buffer de entrada del UART, los leemos y los
         chequeamos para la respuesta.
18   if(Serial1.available() > 0)
19   {
20   SMS[x] = Serial1.read();
21   x++;
22
23   // Comprueba si la respuesta deseada (OK) es la respondida por el módulo.
24   if (strstr(SMS, "OK") != NULL){answer = 1;}
25   }
26   }while(answer == 0); // Esperamos por la respuesta del módulo con el time
         out.
27   SMS[x] = '\0';
28   Serial.print(SMS);
29   i=0;
30   x=0;
31   do
32   {
33   if (SMS[i]=='\r'){SMS[i]=',';}
34   if (SMS[i]=='\n'){SMS[i]=' ';}
35   i++;
```

```
36      }while(i<=101);
37      descomponerSms();
38      i=0;
39      x=0;
40      if (strcmp(smsRecibido," OP")==0 && Estado==1 && sat >=3)
41      {
42      EnviaComandoAT("AT+CMGD=1,4","OK",1000);
43
44      if (numSel==1){establecer_numero_tlf (numero_Tlf_1,Numero_PIN);} //
45      if (numSel==2){establecer_numero_tlf (numero_Tlf_2,Numero_PIN);} //
46      if (numSel==3){establecer_numero_tlf (numero_Tlf_3,Numero_PIN);} //
47      envia_mensaje_gsm (Array_Hora);          // Enviamos el SMS.
48      envia_mensaje_gsm (Array_Longitud_sms); // Enviamos el SMS.
49      envia_mensaje_gsm (Array_Latitud_sms);  // Enviamos el SMS.
50      finaliza_envio_gsm ();
51      }
52      }
53      else
54      {
55      Serial.print("error ");
56      Serial.println(answer, DEC);
57      }
58  }
```

16.2. Compatibilidad del programa fuente con Arduino UNO

El programa fuente que se presenta en el capítulo 14 tendría que ser a priori totalmente compatible con Arduino UNO, dado que en la sección 19, todos los códigos fuente que en esa sección se presentan son totalmente compatibles tanto para Arduino Mega como para Arduino UNO, excepto el programa de la sección 19.5, que, sin realizarle modificaciones solo es valido para Arduino Mega.

Han sido muchas las alternativas tomadas para que el programa final pudiese ser implementado con Arduino UNO, las modificaciones que deben hacerse en el programa que presenta el capítulo 14 para que funcione correctamente en Arduino UNO son numerosas y la mayoría de ellas pasan por comentar lineas de código del programa quitando funciones como el cambio de hora o algún modo de funcionamiento, por tanto en este apartado no indicaremos todas esas modificaciones, dado que le afecta a muchas secciones y tendríamos que volver a colocar todo el código de nuevo.

Además, durante la implementación del código en Arduino MEGA nos hemos dado cuenta de que el micro-controlador que posee Arduino UNO dispone de un tamaño de pila o stack inferior al de Arduino MEGA, si en el programa fuente realizado para Arduino Mega estamos utilizando un tamaño de pila superior a 8, el programa no será válido para Aduino UNO.En este caso, el compilador de Arduino no dará errores, pero el programa no funcionará correctamente.

Otro de los errores que se puede estar produciendo es el de los “saltos a subrutinas”, este

error se produce, por ejemplo, (hablando desde el punto de vista de un microcontrolador), si realizamos una llamada a una función situada 200 posiciones de memoria por encima o por debajo de la posición actual, siendo el salto máximo de 150 posiciones de memoria. Para saber si es este el error que se esta produciendo es necesario consultar el datasheet del microcontrolador que implementa Arduino UNO.

Ya hemos comentado en el apartado de “Análisis de las soluciones” que Arduino UNO solo cuenta con dos interrupciones externas, con lo cual tendríamos que utilizar una entrada digital sobrante para conectar el botón de selección, utilizaríamos el pin 7 utilizando la técnica “polling”.Si pensamos que uno de los errores por los cuales el programa fuente de Arduino MEGA no es compatible con Arduino UNO es por culpa de la disposición de los pines, estamos equivocados, dado que los cambios realizados son correctos.

En resumen, el programa que se presenta en el capitulo14, debido a problemas con la memoria de Arduino UNO, y aun haciéndole los cambios pertinentes en la disposición de pines en lo que implica al botón de selección (utilizando el pin 7 utilizando la técnica “polling”, en lugar de interrupción 4 conectado al pin 19 de Arduino Mega), no es compatible.

Capítulo 17

Posibles mejoras del presente proyecto y más posibilidades que nos ofrece el módulo GPS utilizado

17.1. Posibles mejoras del presente proyecto

En la presente sección se indican las mejoras que se le pueden realizar al presente proyecto así como otros proyectos u otras posibilidades que nos puede dar el módulo GPS/GPRS/GSM V3.0 utilizado. Estas mejoras son las que se enumeran a continuación:

1. Posibilidad de control total del dispositivo desde la aplicación realizada, cambio de hora, tiempo de envío, posibilidad de cambio de los números de teléfono, posibilidad de elección de los números de teléfono a enviar y poder seleccionar el modo de funcionamiento.

2. Disminuir el tiempo de respuesta del módulo ante una petición de usuario, actualmente es de un minuto.

3. Posibilidad de enviar los datos a más de un solo numero de teléfono simultáneamente.

4. Mejorar la aplicación Android de forma que también nos pueda realizar un seguimiento mas exhaustivo del dispositivo tal como consultar rutas realizadas por el mismo indicándonos el trazado seguido...

17.2. Posibilidades que nos ofrece el módulo GPS utilizado

El módulo utilizado en el presente proyecto no solo nos ofrece la posibilidad de ser utilizado para la localización de objetos puesto que prácticamente se puede utilizar como un teléfono móvil, pues el módulo, además de ser capaz de ubicarse y enviar mensajes, es capaz de realizar y recibir llamadas .

A continuación se procede a citar algún proyecto que se podría realizar con el mismo.

- Sistema de seguridad de viviendas basado en Arduino y el módulo GPS/GPRS/GSM V3.0 donde el módulo seria capaz de enviarnos un SMS indicando si se ha abierto alguna puerta de nuestro domicilio, realizar una llamada en modo escucha para saber lo que puede estar ocurriendo en el interior y dar orden de sacar una foto o grabar al interior .
- Sistema de control de ganado basado en Arduino, este proyecto se realizaría de forma similar al presente, lo único que cambiaría seria que el diseño del circuito fuese mas pequeño de forma que pueda adaptarse a la ergonomía de cualquier animal de gran envergadura y ser transportado con facilidad.
- Teléfono móvil basado en Arduino.Este proyecto se aprovecharía al máximo todos los recursos que posee el módulo.En este proyecto, se incorporaría un teclado para poder marcar el numero a llamar y también para poder enviar un mensaje, un altavoz para poder escuchar las llamadas recibidas y un micrófono para poder comunicarnos.
- Servicio de telegestión de contadores. El módulo interaccionará con algún servidor enviando los datos referentes al consumo de nuestro domicilio para poder ser consultados en la red, de este modo se evitarían los inconvenientes relacionados con la estimación de las facturas.

TÍTULO: **CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO**

PLANOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

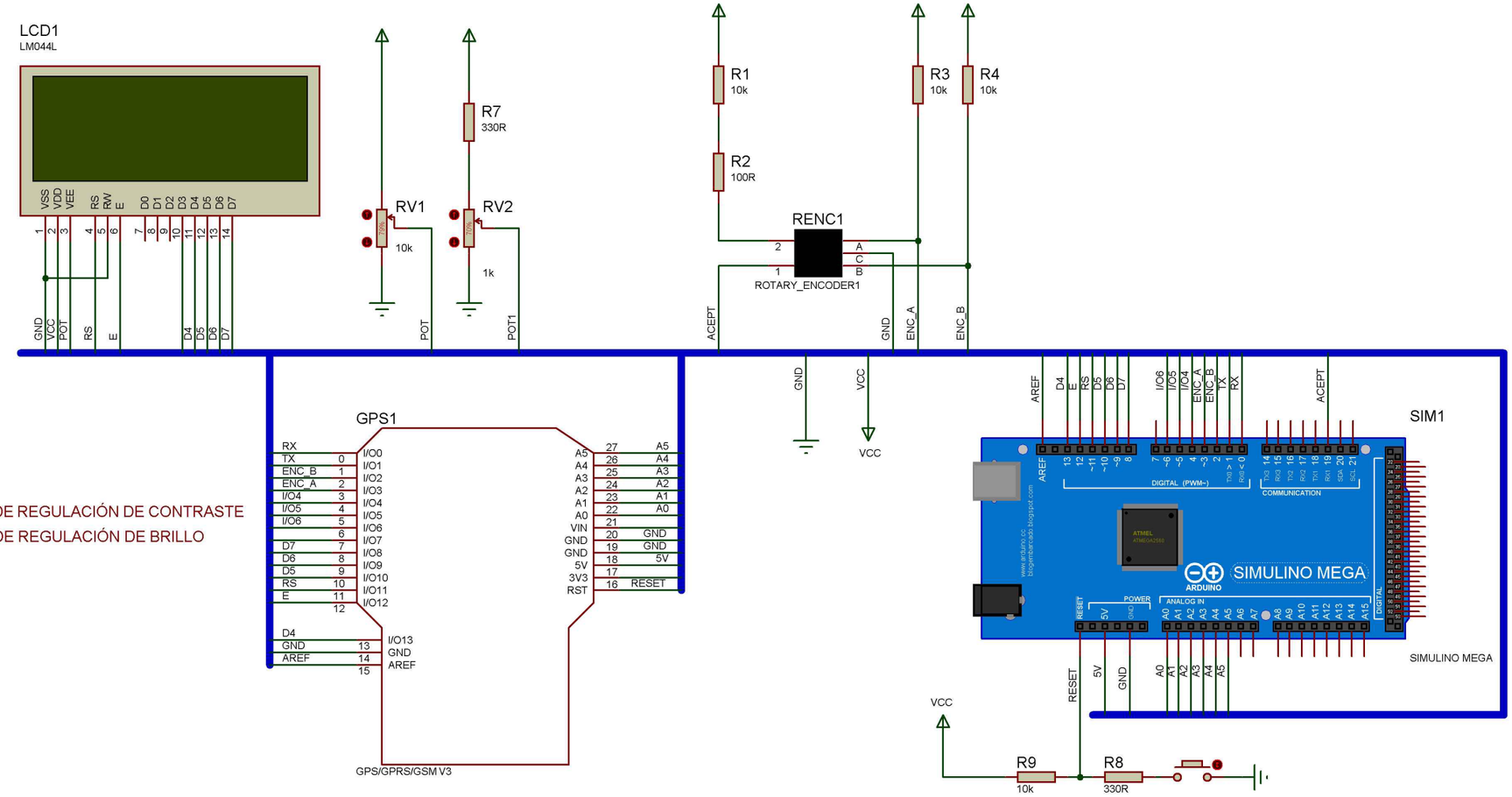
15405 - FERROL

FECHA: JUNIO DE 2015

AUTOR: EL ALUMNO

Fdo.: CRISTÓBAL GARCÍA CAMOIRA

Índice de planos



RV1= POTENCIOMETRO DE REGULACIÓN DE CONTRASTE

RV2= POTENCIOMETRO DE REGULACIÓN DE BRILLO

UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

PLANO GENERAL DE MONTAJE

FECHA: JUNIO 2015

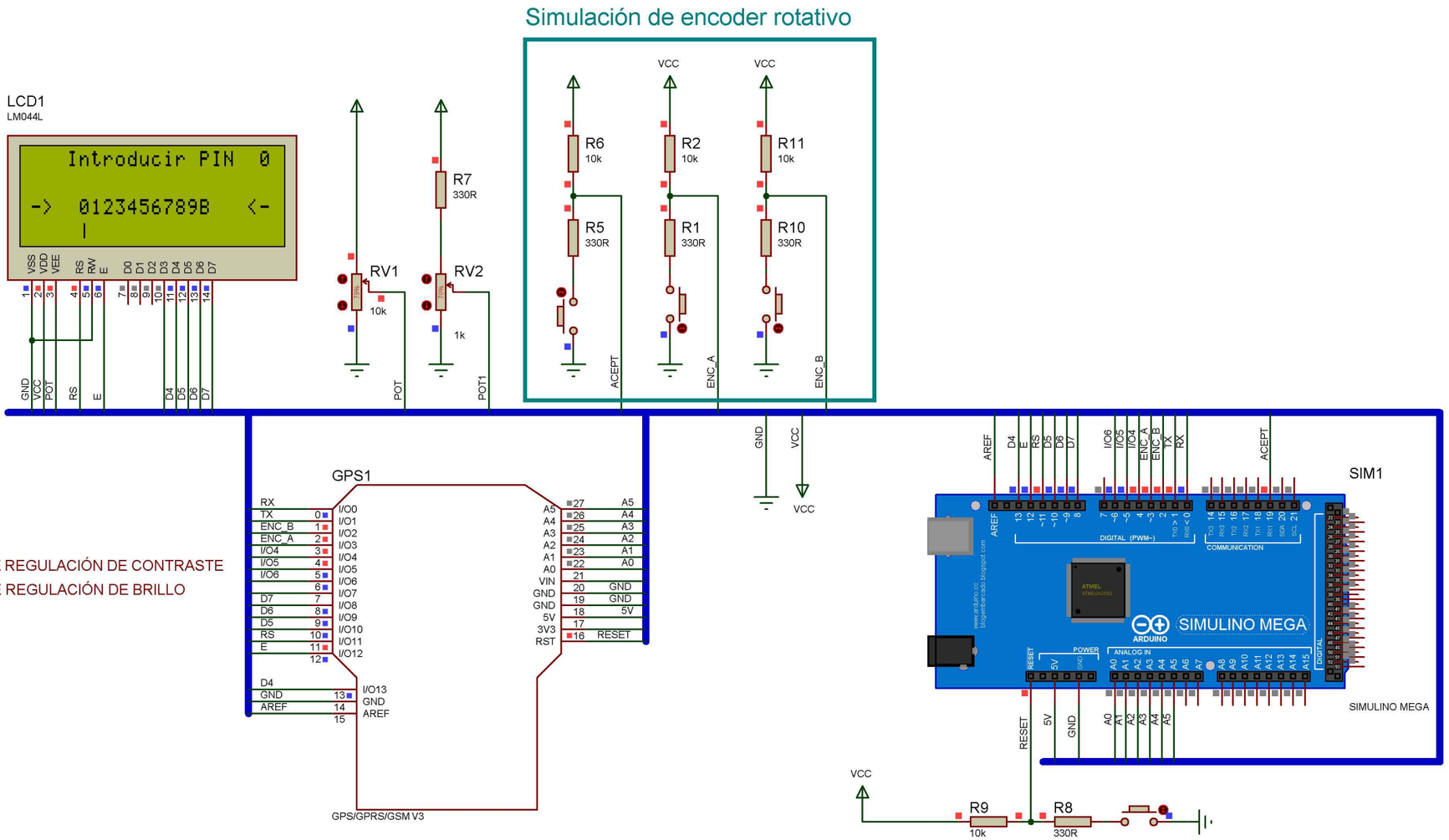
ESCALA: ---

PLANO Nº: 01

AUTOR: CRISTÓBAL GARCÍA CAMOIRA

FIRMA:

A3



RV1= POTENCIOMETRO DE REGULACIÓN DE CONTRASTE

RV2= POTENCIOMETRO DE REGULACIÓN DE BRILLO

UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

PLANO DE SIMULACION EN PROTEUS

FECHA: JUNIO 2015

ESCALA: ---

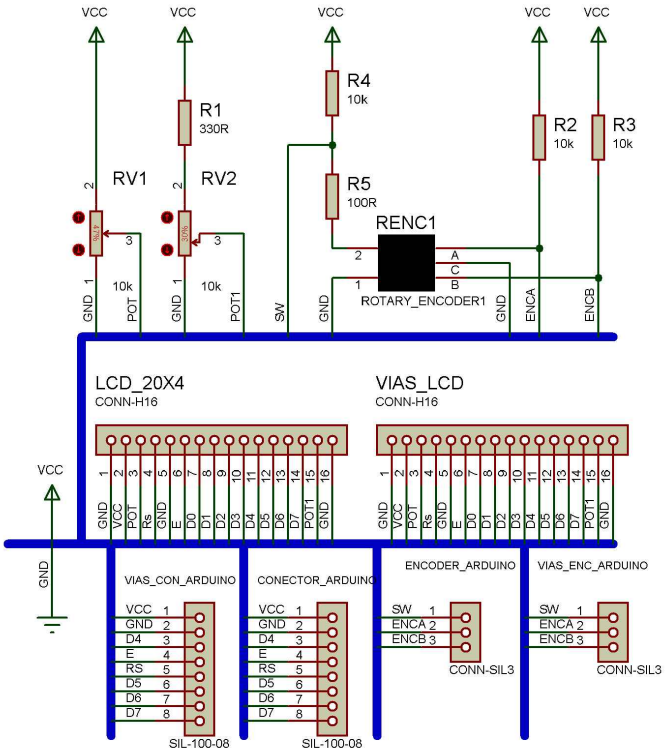
AUTOR:

CRISTÓBAL GARCÍA CAMOIRA

FIRMA:

PLANO Nº: 02

A3



UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

DISEÑO DE PLACA DE CONTROL LCD Y ENCODER

FECHA: JUNIO 2015

ESCALA: ---

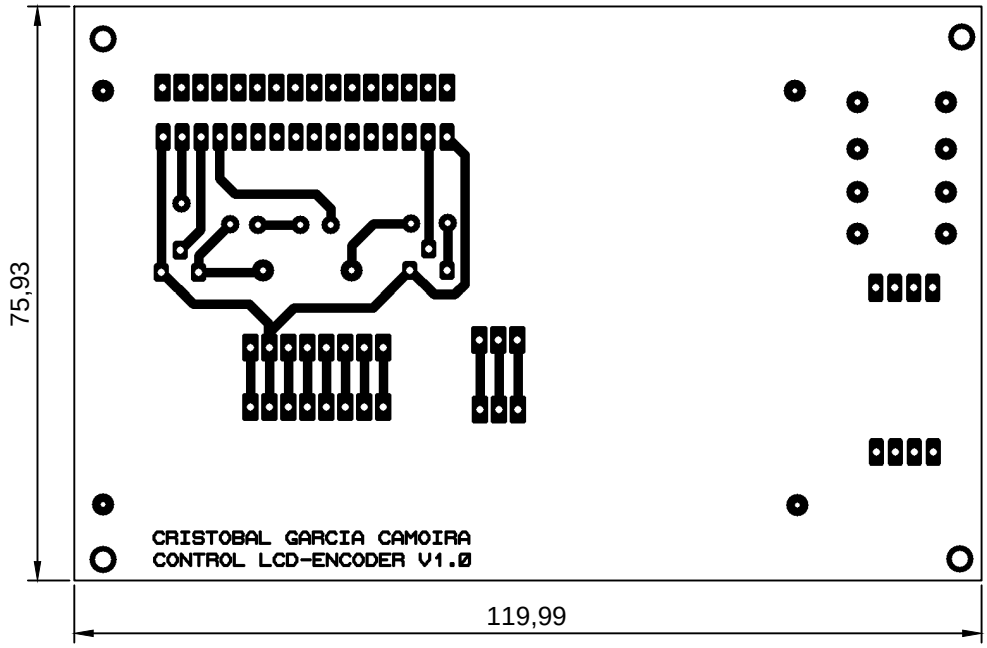
AUTOR:

CRISTÓBAL GARCÍA CAMOIRA

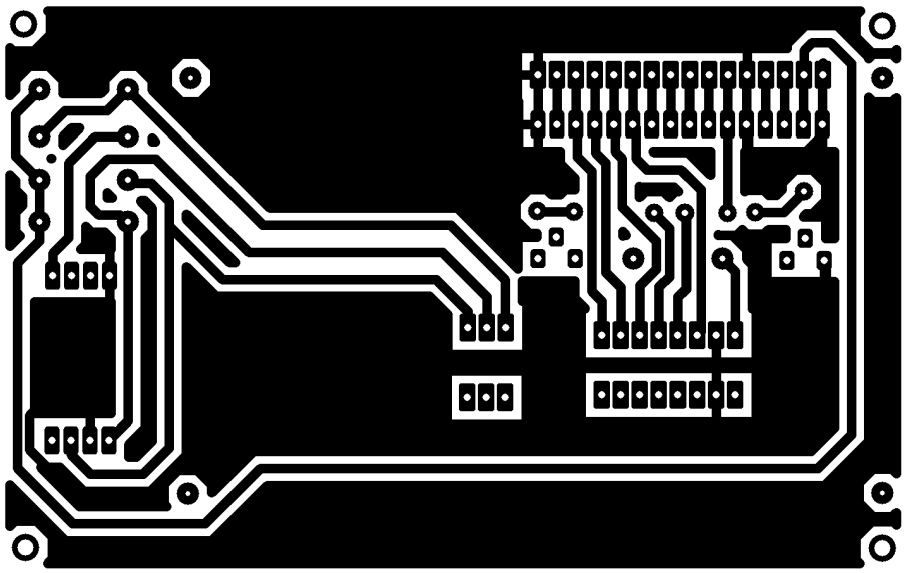
FIRMA:

PLANO Nº: 03

Pistas cara de arriba



Pistas por la cara trasera en espejo



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

PLACA DE CONTROL DE LCD Y ENCODER (PISTAS 1)

FECHA: JUNIO 2015

ESCALA: 1:1

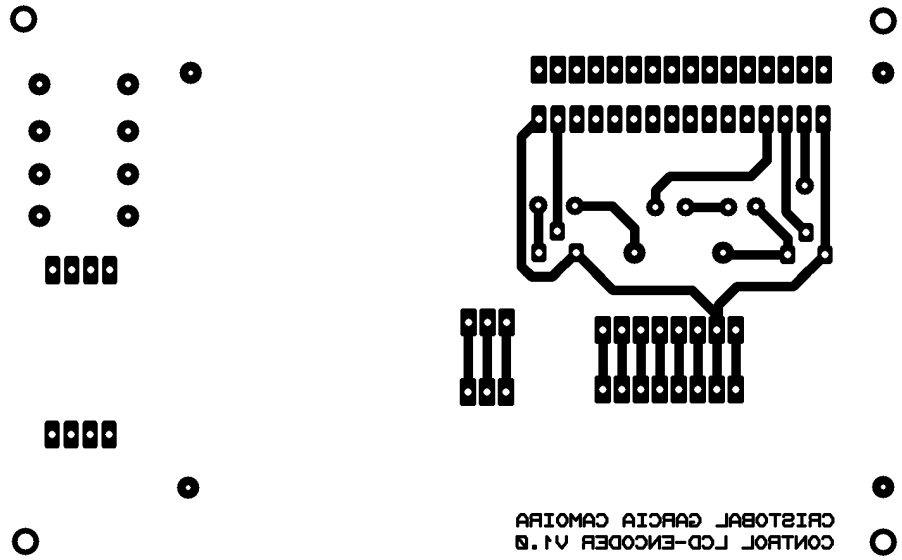
AUTOR:

CRISTÓBAL GARCÍA CAMOIRA

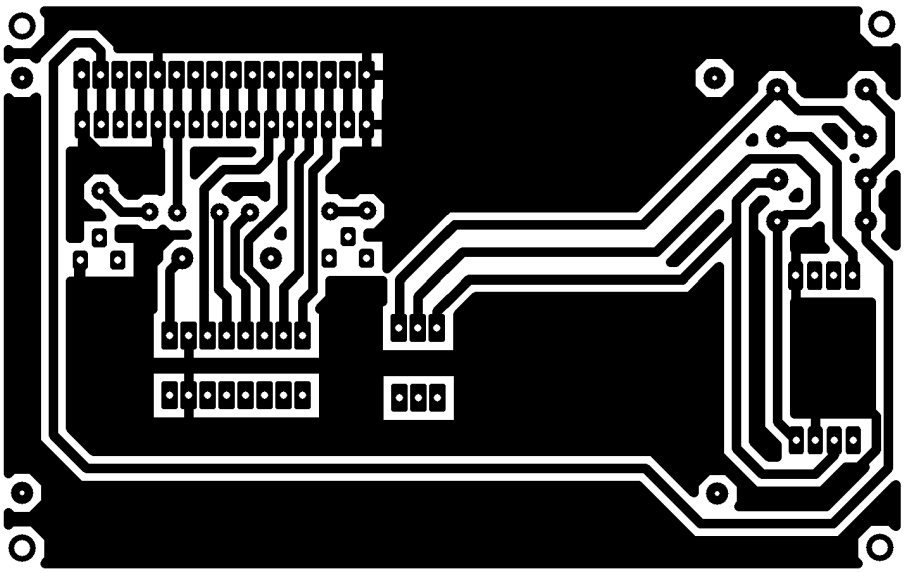
FIRMA:

PLANO Nº: 04

Pistas por la cara delantera en espejo



Pistas por la cara trasera



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

PLACA DE CONTROL DE LCD Y ENCODER (PISTAS 2)

FECHA: JUNIO 2015

ESCALA: 1:1

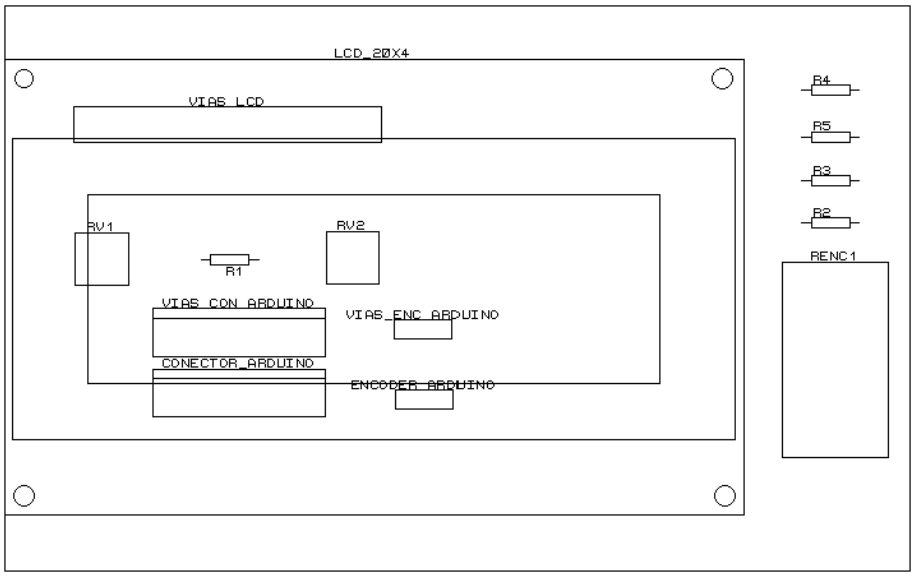
AUTOR:

CRISTÓBAL GARCÍA CAMOIRA

FIRMA:

PLANO Nº: 05

Vista de la cara de los componentes



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

PLACA DE CONTROL DE LCD Y ENCODER (COMPONENTES)

FECHA: JUNIO 2015

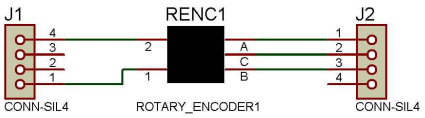
ESCALA: 1:1

AUTOR:

CRISTÓBAL GARCÍA CAMOIRA

FIRMA:

PLANO Nº: 06



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

PLACA DE INSERCION DEL ENCODER ROTATIVO

FECHA: JUNIO 2015

ESCALA: ---

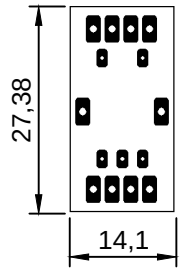
AUTOR:

CRISTÓBAL GARCÍA CAMOIRA

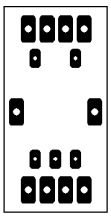
FIRMA:

PLANO Nº: 07

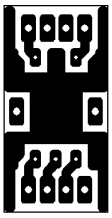
Pistas por la cara delantera en espejo



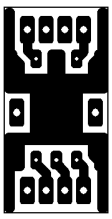
Pistas por la cara delantera



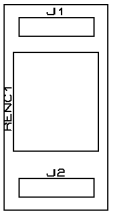
Pistas por la cara trasera



Pistas por la cara trasera en espejo



Vista de la cara de los componentes



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

PLACA DE INSERCION DEL ENCODER ROTATIVO (PISTAS)

FECHA: JUNIO 2015

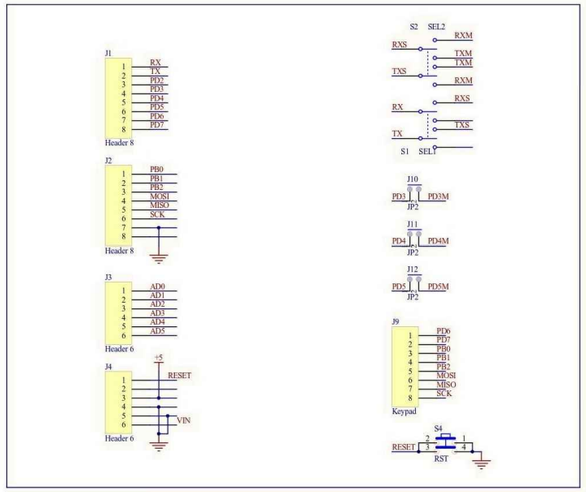
ESCALA: 1:1

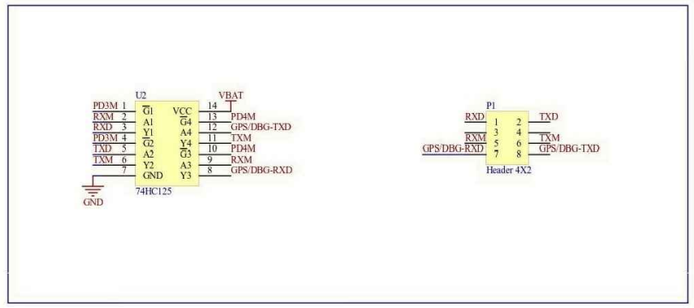
AUTOR:

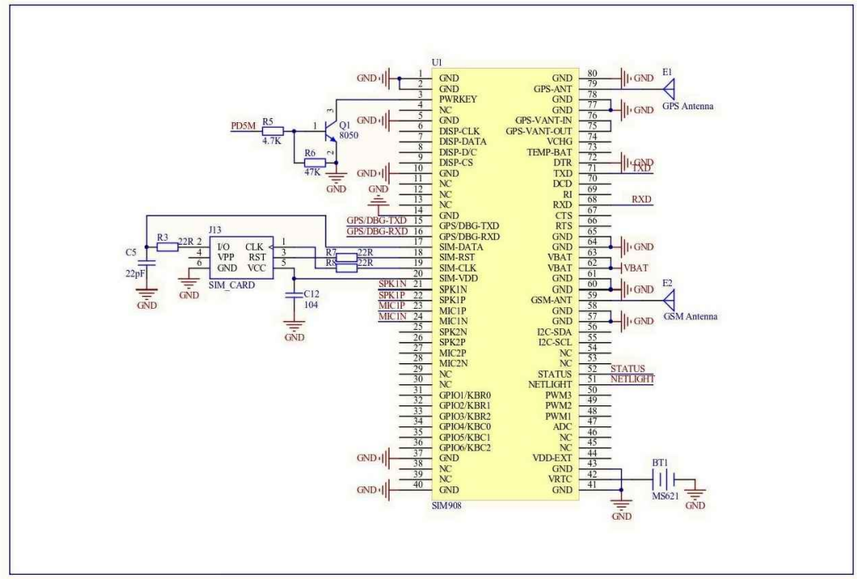
CRISTÓBAL GARCÍA CAMOIRA

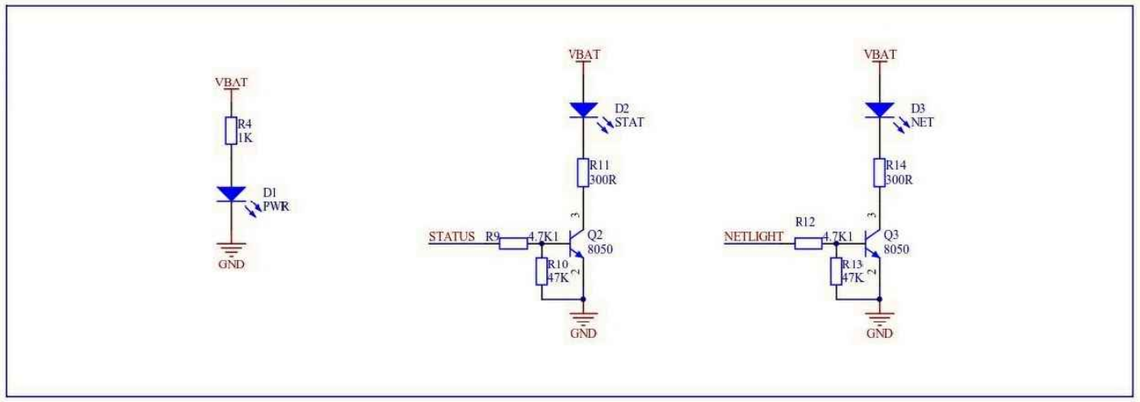
FIRMA:

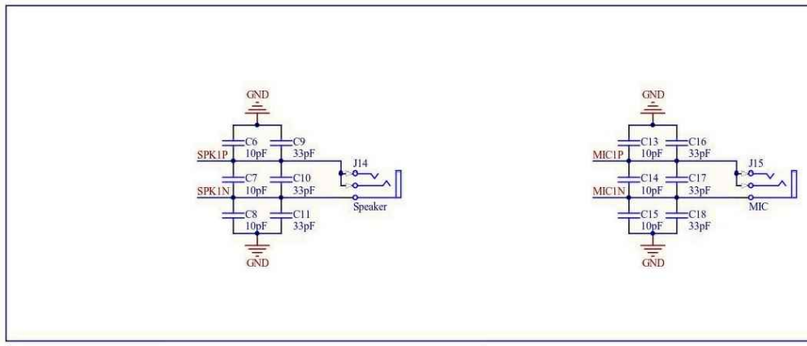
PLANO Nº: 08

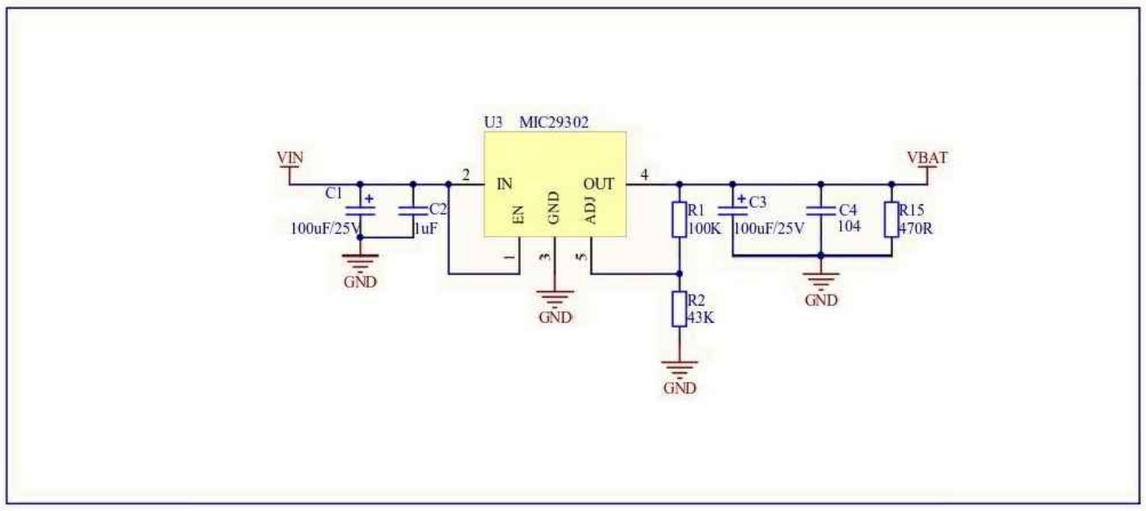












UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

PLANO DE CONSTRUCCIÓN DEL MÓDULO GPS

FECHA: JUNIO 2015

ESCALA: ---

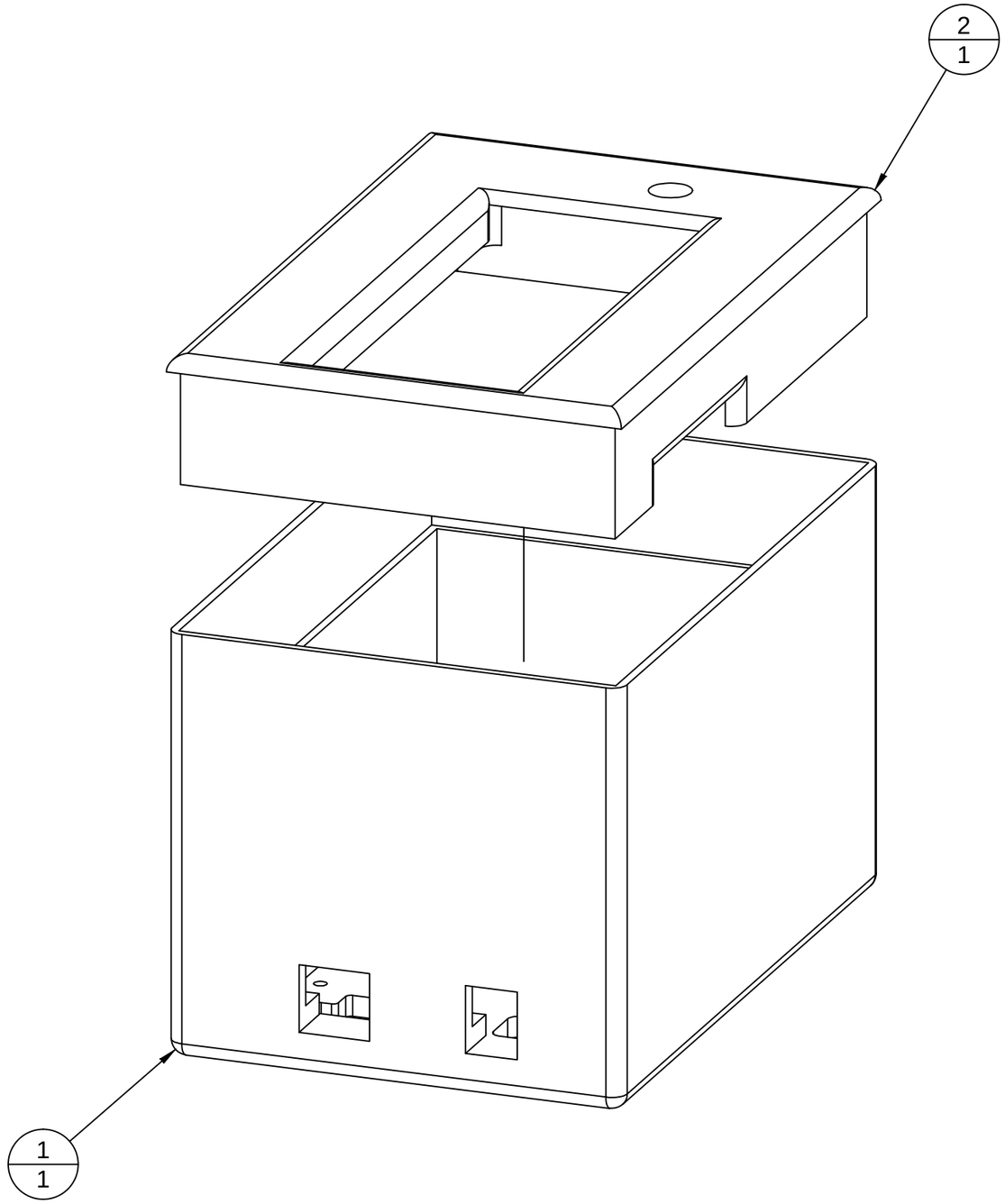
AUTOR:

CRISTÓBAL GARCÍA CAMOIRA

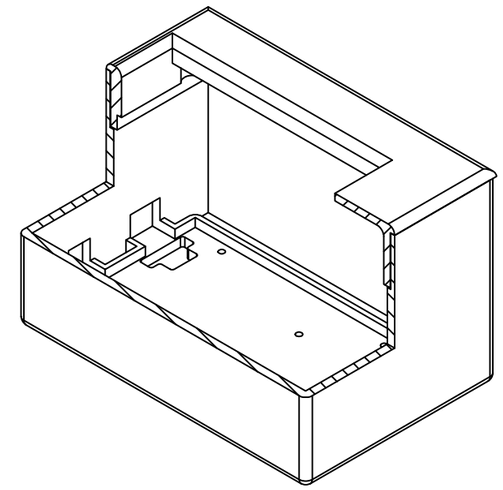
FIRMA:

PLANO Nº: 09

A3



Numero de pieza	File Name (no extension)	Autor	Cantidad
1	BASE_GPS	CRISTOBAL	1
2	CARCASA_SUPERIOR	CRISTOBAL	1



1:2

UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

VISTA DEL CONJUNTO DE LA CAJA DEL LOCALIZADOR

FECHA: JUNIO 2015

ESCALA: 1:1

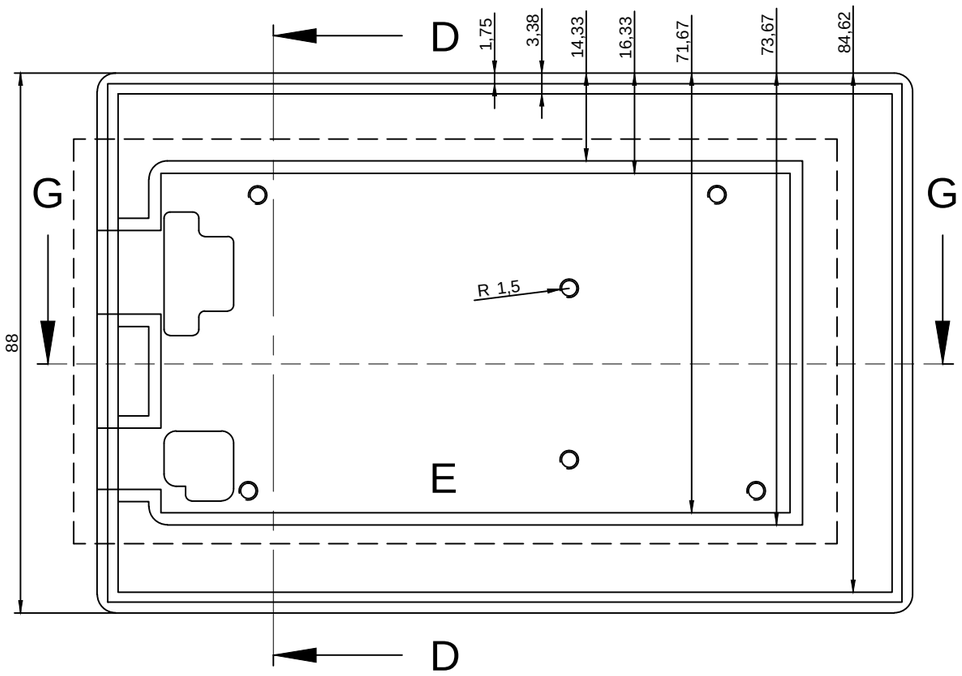
AUTOR:

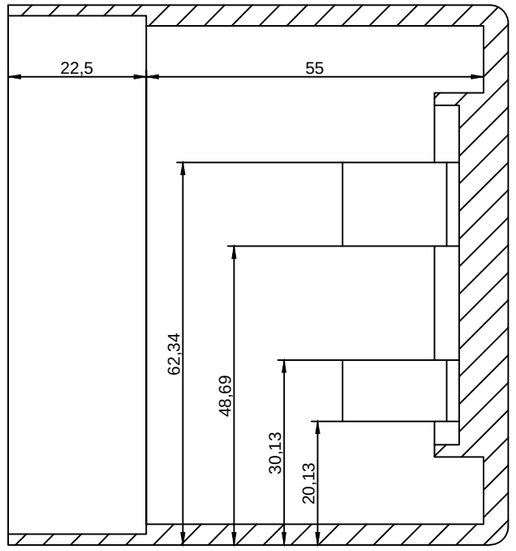
CRISTÓBAL GARCÍA CAMOIRA

FIRMA:

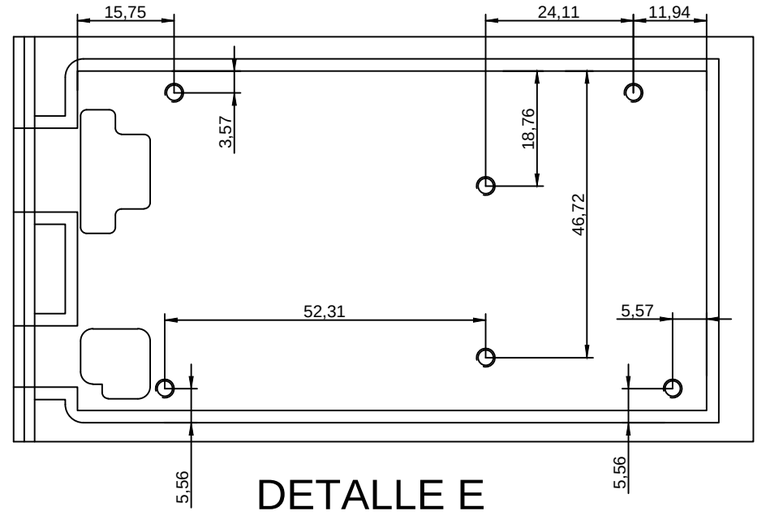
PLANO Nº: 10

A3

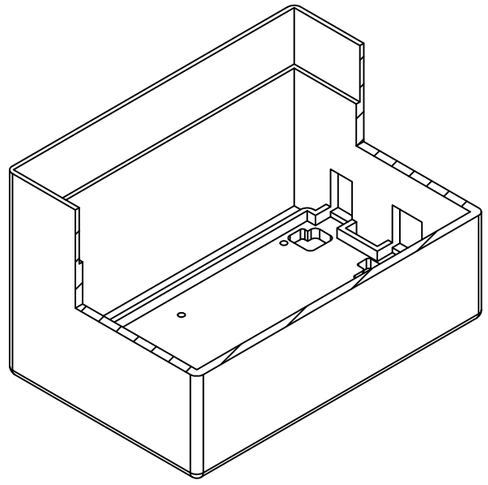




SECCION D-D

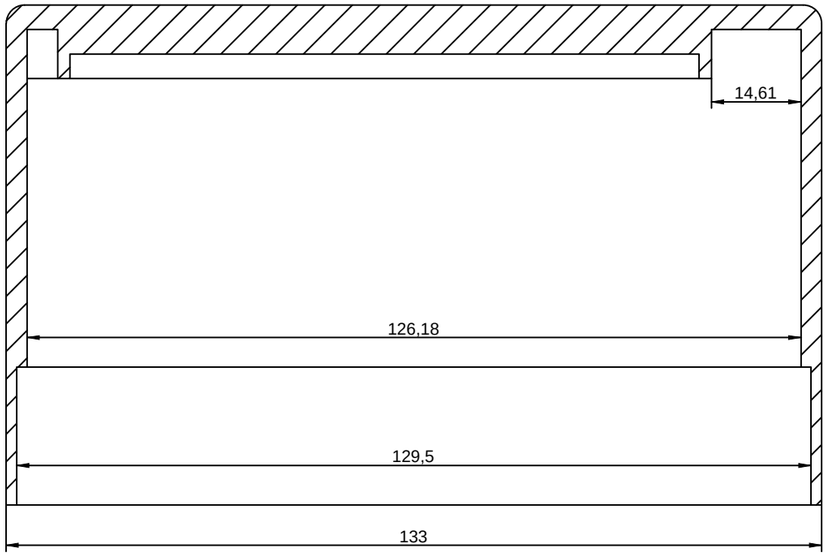


DETALLE E



1:2

SECTION G-G



UNIVERSIDADE DA CORUÑA ESCUELA UNIVERSITARIA POLITÉCNICA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TFG Nº: 770G01A88

TÍTULO DEL TFG:

CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO

TÍTULO DEL PLANO:

PLANO DE LA BASE DE LA CAJA DEL LOCALIZADOR

FECHA: JUNIO 2015

ESCALA: 1:1

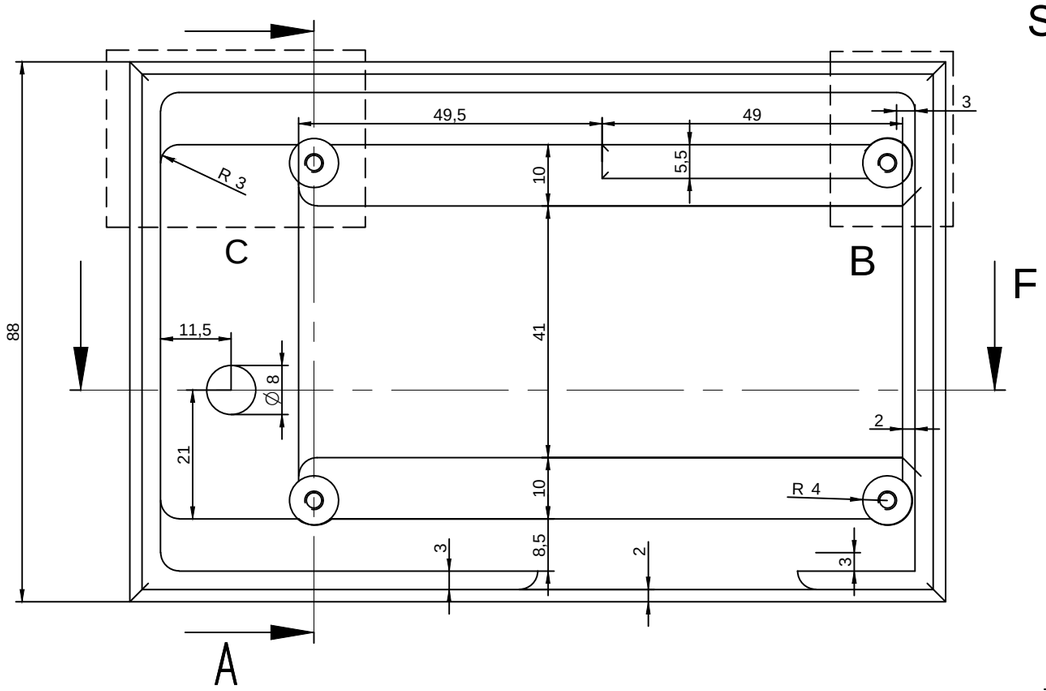
AUTOR:

CRISTÓBAL GARCÍA CAMOIRA

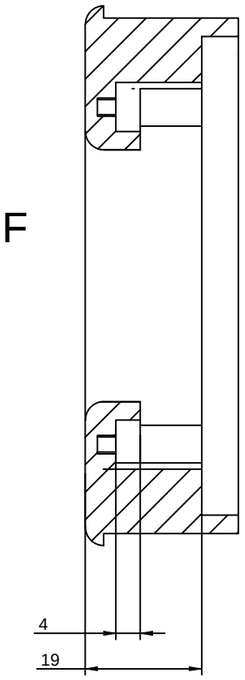
FIRMA:

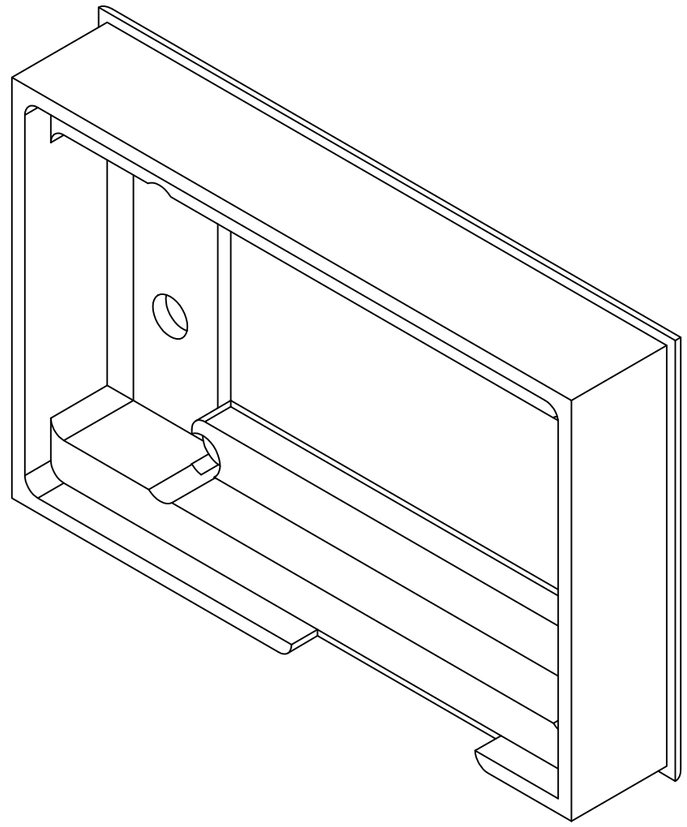
PLANO Nº: 11

A3

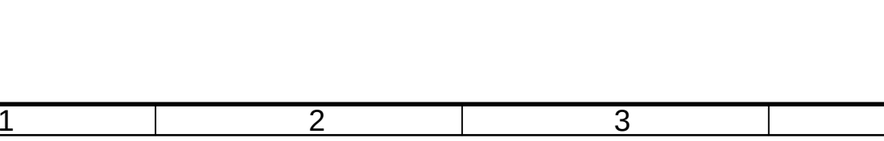


SECCIÓN A-A

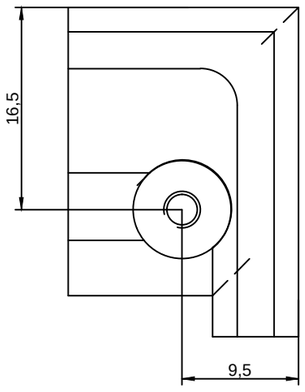




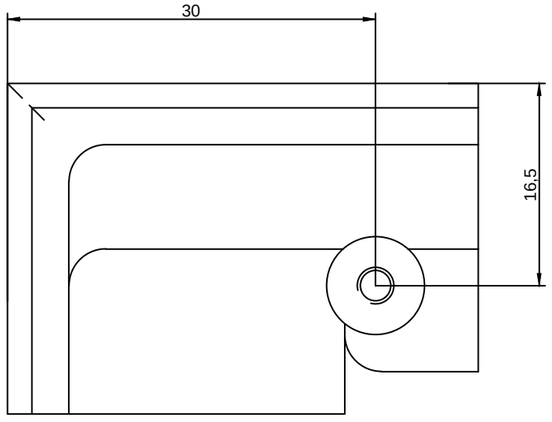
1:1



SECCION F-F



DETAIL B



DETAIL C

UNIVERSIDADE DA CORUÑA — ESCUELA UNIVERSITARIA POLITÉCNICA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA	TFG Nº: 770G01A88
TÍTULO DEL TFG: CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO	
TÍTULO DEL PLANO: PLANO DE LA TAPA DE LA CAJA DEL LOCALIZADOR	FECHA: JUNIO 2015
	ESCALA: 1:1
AUTOR: CRISTÓBAL GARCÍA CAMOIRA	FIRMA:
	PLANO Nº: 12

TÍTULO: **CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO**

PLIEGO DE CONDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2015

AUTOR: EL ALUMNO

Fdo.: CRISTÓBAL GARCÍA CAMOIRA

Índice del pliego de condiciones

Capítulo 18

Especificaciones de los materiales

Los materiales y componentes utilizados en la realización del presente proyecto han sido aquellos que reunieran los requisitos de diseño. Además éstos debían asegurar un correcto funcionamiento durante la vida del dispositivo. El subministrador debe garantizar el funcionamiento correcto de cada componente según las hojas características del mismo.

Capítulo 19

Pruebas de verificación

A continuación se detallan las pruebas de verificación necesarias para comprobar el correcto funcionamiento de cada uno de los módulos que componen el control de posicionamiento.

19.1. Programa Arduino

Para la realización de las pruebas de verificación se ha utilizado la versión IDE 1.0.5 instalable de Arduino, todos los programas descritos a continuación y realizados para poder verificar el funcionamiento de los componentes están basados en esta versión y comprobados prácticamente, por tanto no se puede asegurar el funcionamiento de los mismos programas cargados a Arduino desde versiones del programa tanto anteriores como posteriores.

19.2. LCD

Se debe comprobar que el LCD enciende correctamente con tan sólo introducirle la tensión de alimentación. Se recomienda realizar pruebas de que funcionan las cuatro líneas disponibles. Para ello, conectaremos nuestro LCD de la forma que nos indica el plano [1] a nuestro Arduino, ya sea UNO, LEONARDO o MEGA. Cargaremos el siguiente programa en el entorno de programación de Arduino el cual muestra en las cuatro líneas del LCD “Hola Mundo”.

```
1  //*****************************************************************
2  // Autor : CRISTÓBAL GARCÍA CAMOIRA
3  //*****************************************************************
4
5  #include <LiquidCrystal.h>
6  LiquidCrystal lcd(11, NULL, 12, 13, 10, 9, 8);
7
8  void setup()
9  {
10 lcd.begin(20,4);
11 }
12
13 void loop()
```

```
14 {
15 lcd.setCursor(0,0);
16 lcd.print("Hola Mundo");
17 lcd.setCursor(0,1);
18 lcd.print("Hola Mundo");
19 lcd.setCursor(0,2);
20 lcd.print("Hola Mundo");
21 lcd.setCursor(0,3);
22 lcd.print("Hola Mundo");
23 }
24 //*****************************************************************
```

NOTA: Si por algún error del montaje no se muestra nada en la pantalla, verifica el conexionado de los potenciómetros de contraste y brillo, es un error muy común.

19.3. Arduino

Debe comprobarse que el Arduino utilizado está en perfectas condiciones, se recomienda probar las salidas que se van a utilizar. También se debe comprobar el regulador de tensión de 5v de Arduino.

19.4. GPS GPRS GSM

Para la comprobación del funcionamiento del módulo GPS, es necesario poseer además de dicho módulo, un Arduino, preferiblemente Arduino UNO o MEGA . El conexionado de dicho módulo con la placa Arduino UNO se realiza de forma intuitiva, dado que la disposición de los pines de dicho módulo hacen que solo exista una única posibilidad de insertarlo, en el caso de utilizar Arduino MEGA, debemos recurrir al plano [1].

Con el siguiente programa,cargado desde la interfaz a nuestro Arduino podremos comprobar la recepción de las tramas de nuestro módulo, ya sea visualizándolas con el monitor serial propio de la interfaz de Arduino, o bien mediante la utilización de nuestro Hyperterminal, una aplicación de Windows que le permite establecer una comunicación ordenador a ordenador o a cualquier otro dispositivo a través de una conexión telefónica convencional o por puerto serial.

```
1
2 //*****************************************************************
3 // GPS/GPRS/GSM Module V3.0
4 //*****************************************************************
5 // Autor : CRISTÓBAL GARCÍA CAMOIRA
6 //*****************************************************************
7 // # Descripción:
8 // # El presente programa sirve para probar el funcionamiento de
9 // # nuestro GPS con la placa de nuestro Arduino.
10 // # Pasos:
```

```
11 // # 1. Ponemos el interruptor S1 en el modo Prog (A la derecha).
12 // # 2. Ponemos el interruptor S2 en el modo Arduino(A la izquierda).
13 // # 3. Ponemos el interruptor UART en el centro de sus tres
14 // #    posibles posiciones.
15 // # 4. Subir el programa al Arduino.
16 // # 5. Poner el interruptor s1 hacia la izquierda en el modo comm (left side)
17 // # 6. Resetear el aparato.(RST)
18 // # wiki link- http://www.dfrobot.com/wiki/index.php/GPS/GPRS/GSM_Module_V3.0_(SKU:
       TEL0051)
19
20 //Incluimos librerías
21
22 #include <stdio.h>
23 #include <stdlib.h>
24
25 #define TAMANO_ARRAY 200 // Tamaño de buffer
26
27 #define habilitar_gps()       digitalWrite (4, LOW)
28 #define deshabilitar_gps()    digitalWrite (4, HIGH)
29
30 #define habilitar_gsm()       digitalWrite (6, LOW)
31 #define deshabilitar_gsm()    digitalWrite (6, HIGH)
32
33  /* Variables del programa*/
34  byte byteGPS = 0;              // Donde guardaremos cada byte que leamos del GPS
       para mandarlos al array.
35  int i = 1;                     // Variable que utilizaremos como acumulador,
       inicializamos su valor en 1.
36  char TramaGPG[TAMANO_ARRAY];   // Array donde iremos almacenando los bytes leídos
       del GPS.
37
38 void setup()
39 {
40   Serial.begin(9600);
41   gps_inicio_pines () ;
42   comenzar_gps () ;
43  }
44
45 void loop()
46 {
47   leer_gps ();
48 }
49
50 void comenzar_gps () {
51   digitalWrite(5,HIGH);
52   delay(1500);
53   digitalWrite(5,LOW);
54   delay(1500);
55   habilitar_gsm ();
```

```
56    deshabilitar_gps ();
57    delay(2000);
58    Serial.println("AT");
59    delay(1000);
60    Serial.println("AT+CGPSPWR=1");
61    delay(1000);
62    //Resetea el GPS  en modo autónomo.
63    Serial.println("AT+CGPSRST=1");
64    delay(1000);
65    habilitar_gps();
66    deshabilitar_gsm();
67    delay(2000);
68 }
69
70 //
71 void leer_gps ()
72 {
73     if (Serial.available()>0)
74    {
75        byteGPS = Serial.read();
76        // Si el carácter leído es distinto a $ comenzamos a guardar
77        // en el array.
78        // y lo almacenaremos en la cadena de caracteres en la
79        // posición que corresponde incrementamos en uno el contador.
80        if (byteGPS != '$')
81        {
82        TramaGPG[i]=byteGPS;
83        i++;
84        }
85        if (byteGPS == '$')
86        {
87        // En la posición 0 introducimos el símbolo del dólar
88        // para acotar la trama desde el principio.
89        TramaGPG[0]='$';
90        Serial.println();
91        Serial.print(TramaGPG);
92        memset(TramaGPG, 0, sizeof(TramaGPG)); // Inicializa a cero la cadena para
              eliminar
93                                               // restos no deseados de lecturas
                                                    anteriores
94        i=1; // Reseteamos la variable i;
95        Serial.println();
96        }
97     }
98 }
99
100 // Establece los pines de trabajo del módulo GPS
101 void gps_inicio_pines ()
102 {
```

```
103     pinMode (6, OUTPUT);
104     pinMode (4, OUTPUT);
105     pinMode (5, OUTPUT);
106 }
107 //*********************************************************************
```

NOTA: Cabe comentar que se ha hecho una modificación en el conexionado de nuestro módulo GPS a la placa Arduino. Los pines por defecto del Arduino para controlar nuestro módulo GPS se enumeran a continuación:

- Pin 3 de nuestro Arduino : Habilita o deshabilita la conexión GSM.
- Pin 4 de nuestro Arduino : Habilita o deshabilita la conexión GPS.
- Pin 5 de nuestro Arduino : Enciende y apaga el módulo GPS GSM GPRS V3.0

Dado que deseamos hacer compatible el programa para los Arduinos más básicos y dado que en Arduino UNO o LEONARDO utilizamos las interrupciones 0 y 1, situadas en los pines 2 y 3 de Arduino UNO o 3 y 2 en Arduino LEONARDO , tema del cual ya hemos hablado en el apartado Análisis de las soluciones, necesitamos el pin 3 libre para poderlo utilizar como interrupción.

Esas conexiones del módulo GPS a nuestro Arduino están realizadas por medio de 3 jumpers (j10,j11,j12) para ello visualizaremos el plano [9], donde se indica que el jumper que une el pin 3 del Arduino con el pin que habilita o deshabilita la conexión GSM de nuestro módulo es el j10. Una vez identificado el jumper que afecta a nuestro montaje, los pasos que se deben seguir para poder controlar la habilitación o deshabilitación de la conexión GSM de nuestro módulo GPS con un pin distinto del que viene por defecto (pin 3 de nuestro Arduino) a cualquier otro de los pines digitales libres del Arduino (en este caso utilizaremos el pin 6), son los siguientes:

1. Extraer el jumper que afecta a la conexión (J10).
2. Unir el pin más próximo al borde de la placa de nuestro módulo GPS con el pin 6 de nuestro Arduino.
    - Esta unión se ha realizado soldando a un conector hembra un cable rígido por uno de sus extremos, y por el otro extremo, uniéndolo al pin 6 de nuestro Arduino,tal y como figura en la siguiente imagen.
3. Podemos actuar de la misma forma con los dos pines restantes, extrayendo sus respectivos jumpers, en el caso de tener ya ocupados los pines 4 y 5 del Arduino.

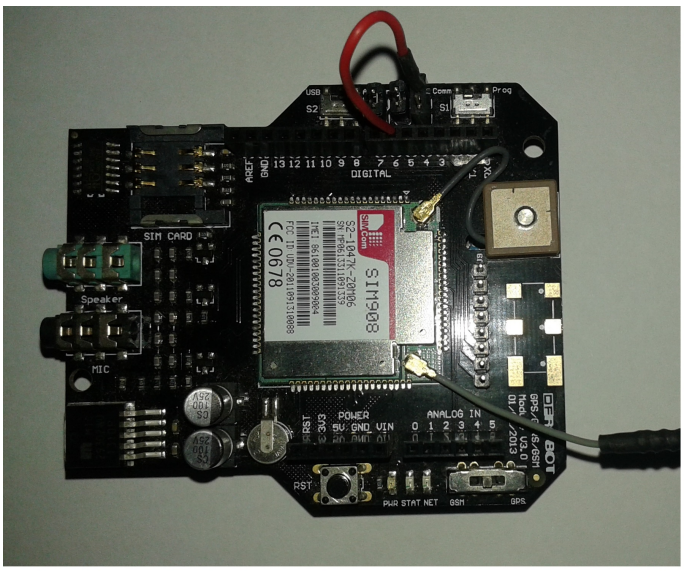


Figura 19.4.0.1 – Conexionado de control de la placa GPS

19.5. Encoder

Para verificar el correcto funcionamiento del encoder rotativo, realizamos el montaje que se indica en el plano [1], y de esta forma además de realizar las pruebas a nuestro encoder, ya tendremos resuelto el montaje definitivo. El siguiente programa nos muestra en la primera linea del LCD el numero de avances de nuestro encoder , tanto a derechas como a izquierdas , incrementándose el contador en el caso de que se gire hacia derechas y decrementando el contador en el caso de girar a izquierdas. Si presionamos el pulsador que incluye nuestro encoder, reseteamos el contador.

```
1
2  //****************************************************************
3  // Autor : CRISTÓBAL GARCÍA CAMOIRA
4  //****************************************************************
5
6  // ******** Rutina para nuestro menú con encoder rotativo ***********
7  // El codificador rotativo genérico tiene tres pines, visto desde el
8  // frente: ACB Giro a la derecha A (encendido) -> B (encendido) ->
9  // A (apagado) -> B (apagado) Contador CW rotación B (encendido) ->
10 // A (encendido) -> B (apagado) -> A (apagado).
11 // Puede también tener 2 pines a mayores siendo estos un interruptor
12 // conectado en este caso al pin 19 de nuestro Arduino MEGA.
13 //****************************************************************
14
15 #include <Wire.h>
16 #include <LiquidCrystal.h>
17 #include <TimerOne.h>
18
```

```
19 LiquidCrystal lcd(11, NULL, 12, 13, 10, 9, 8);
20
21 enum PinAssignments
22 {
23   encoderPinA = 2,    // Derecha (Llamado DT en nuestro encoder)
24   encoderPinB = 3,    // Izquierda (Llamado CLK en nuestro encoder)
25   clearButton = 19,    // Switch (Llamado SW en nuestro encoder)
26 // Conectar +5v y Gnd en los pines restantes.
27 };
28
29 volatile unsigned int encoderPos = 0;       // Contador para el dial.
30 volatile unsigned int lastReportedPos = 1; // Gestión para el cambio.
31 static boolean rotating=false;              // Eliminación de rebotes.
32
33 // Variables para el servicio de la rutina de interrupción.
34 boolean A_set = false;
35 boolean B_set = false;
36
37 void setup()
38 {
39   lcd.begin(20,4);
40   Timer1.initialize(150); // Inicialización del timer3.
41 // Nuevo método para habilitar las resistencias de pullup.
42   pinMode(encoderPinA, INPUT_PULLUP);
43   pinMode(encoderPinB, INPUT_PULLUP);
44   pinMode(clearButton, INPUT_PULLUP);
45 // Pin del encoder en la interrupción 0 (pin 2)
46 // La interrupción 0 se encargará de ejecutar la función
47 // doEncoderA cuando ve un cambio sobre el pin A.
48   attachInterrupt(0, doEncoderA, CHANGE);
49 // Pin del encoder en la interrupción 1 (pin 3)
50 // La interrupción 1 se encargara de ejecutar la función
51 // doEncoderB cuando ve un cambio sobre el pin B.
52   attachInterrupt(1, doEncoderB, CHANGE);
53  /* Pin del pulsador en la interrupción 4 (pin 19)*/
54  /* La interrupción 4 se encargara de ejecutar la función seleccionar cuando ve un
       cambio sobre el pin 19.*/
55  attachInterrupt(4, borrar, CHANGE);
56 // Actualiza pantalla cada 0.15 segundos
57   Timer1.attachInterrupt(Actualiza_pantalla);
58   Serial.begin(9600);  // Salida
59 }
60 // Bucle principal, el trabajo se realiza mediante rutinas
61 // de servicio de interrupción, este bucle solo imprime por
62 // el puerto serie.
63 void loop()
64 {
65   lcd.setCursor(0,0);
66   lcd.print(encoderPos,DEC);
```

```
67 }
68
69 // Interrupción en A por cambio de estado.
70 void doEncoderA(){
71 // Antirrebote
72   if ( rotating ) delay (1);
73 // Esperar un poco hasta que se produzca el rebote
74 // Prueba de transición, hubo cambio?
75   if( digitalRead(encoderPinA) != A_set )
76   {
77 // Anti-rebote, una vez más
78     A_set = !A_set;
79 // Ajustar contador +1 si A conduce al B
80     if ( A_set && !B_set ) encoderPos += 1;
81 // No más supresión de rebotes a menos que llegue al loop().
82     rotating = false;
83   }
84 }
85
86 // Interrupción en B por cambio de estado, es idéntico al anterior.
87 void doEncoderB()
88 {
89   if ( rotating ) delay (1);
90   if( digitalRead(encoderPinB) != B_set )
91   {
92   B_set = !B_set;
93 // Ajustar contador -1 si B conduce a A
94   if( B_set && !A_set ) encoderPos -= 1;
95   rotating = false;
96   }
97 }
98
99 // Al pulsar el botón del encoder resetea el contador
100 void borrar()
101 {
102   encoderPos = 0;
103   lcd.clear();
104   lcd.print(encoderPos,DEC);
105 }
106
107 void Actualiza_pantalla(void)
108 {
109     rotating = true;  // Reestablecer el circuito antirrebote
110     if (lastReportedPos != encoderPos)
111     {
112     Serial.print("Index:");
113     Serial.println(encoderPos, DEC);
114     lastReportedPos = encoderPos;
115     }
```

```
116 }
117 //*************************************************************
```

19.6. Enviar SMS por comandos AT

El programa a continuación descrito solo sirve para configurar los pines de control del módulo GPS/GPRS/GSM V3.0 con nuestro Arduino MEGA o UNO, se recomienda seguir los pasos que se indican en el mismo para la correcta utilización del módulo.

El montaje para la realización de este apartado es mucho más simple de lo que se refleja en el plano [1], dado que no necesitamos conectar el LCD ni ningún pulsador adicional, solo insertar el módulo GPS en nuestro Arduino, ya sea UNO o MEGA y realizar las modificaciones pertinentes en el módulo GPS tal y como se indica en la sección 19.4.

Se ha de alimentar el dispositivo de la forma que se indica en la sección 21.4 para que funcione correctamente. Tenemos que tener en cuenta que se pueden producir picos de corriente de 2A durante el envío de un SMS.

Este programa ha sido probado tanto en Arduino UNO como en Arduino MEGA.

```
1  //*************************************************************
2  // GPS/GPRS/GSM Module V3.0
3  //*************************************************************
4  // Autor : CRISTÓBAL GARCÍA CAMOIRA
5  //*************************************************************
6  // # Descripción:
7  // # El presente programa sirve para probar el funcionamiento de
8  // # nuestro Módulo GPS GPRS GSM V3.0 mediante el envío de comandos AT.
9  // # Pasos:
10 // # 1. Ponemos el interruptor S1 en el modo Prog (A la derecha).
11 // # 2. Ponemos el interruptor S2 en el modo USB(A la derecha).
12 // # 3. Ponemos el interruptor UART en el centro de sus tres
13 // #    posibles posiciones.
14 // # 4. Subir el programa al Arduino.
15 // # 5. Poner el interruptor s1 hacia la izquierda en el modo comm.
16 // # 6. Resetear el aparato.(RST)
17 // # wiki link- http://www.dfrobot.com/wiki/index.php/GPS/GPRS/GSM_Module_V3.0_(SKU:
       TEL0051)
18
19 void setup()
20  {
21    //Inicializa los pines para funcionar en modo GSM.
22     pinMode(6,OUTPUT);
23     pinMode(4,OUTPUT);
24     pinMode(5,OUTPUT);
25    //Output GSM Timing
26     digitalWrite(5,HIGH);
27     delay(1500);
28     digitalWrite(5,LOW);
```

```
29 }
30 void loop()
31 {
32  // Usar los siguientes comandos en lugar de utilizar el interruptor hardware (UART
       )
33  // para seleccionar si deseamos trabajar en modo GSM o bien en modo GPS.
34  // Si deseamos trabajar con ambos (GSM y GPS) podremos con el código a
       continuación
35  // habilitar o deshabilitar GSM y GPS según nuestras necesidades.
36
37    digitalWrite(6,LOW); // Habilita GSM TX-RX
38    digitalWrite(4,HIGH);// Deshabilita GPS TX-RX
39 }
```

Una vez que hemos cargado el código, tendremos que descargar el programa coolterm del siguiente enlace:

`http://freeware.the-meiers.org/`

El programa que hemos descargado es un terminal serie que funciona de manera muy similar al Hyperterminal, no hemos utilizado Hyperterminal, ni el monitor serial de Arduino, porque no hemos conseguido enviar el terminador de SMS (“1A”) en hexadecimal.

Pasos para la configuración del terminal Coolterm:

1. Abrimos el software Coolterm y seleccionamos “Options” como se muestra en la figura 19.6.0.2.

2. Seleccionamos el puerto serie por el que se ha conectado nuestro Arduino. Si no estamos seguros de la ubicación del puerto serie en el cual se ha conectado nuestro Arduino, podremos verificarlo haciendo click derecho en mi pc o computer, administrar, administrador de dispositivos, puertos COM Y LPT.

3. Debemos asegurarnos de que la velocidad de transmisión se establece en 9600. El número de bits de los datos es 8. Se ajusta paridad a “ninguno”. Se ajusta el número de bits de parada en 1.Este tercer paso se observa en la imagen que muestra la figura 19.6.0.3.

4. En la lista de la parte izquierda de la ventana de opciones, haga click en “Terminal”. Asegúrese de que la opción “Local Echo” está activada. Esto le permitirá ver lo que esta escribiendo en el terminal. Esto se muestra en la figura 19.6.0.4.

5. Hacemos click en “OK” para grabar la configuración y cerramos la ventana de Opciones.

6. Hacemos click en “Connect ”en la barra de menús.

7. Debemos ver el estado “Connected ”en la barra de estado.

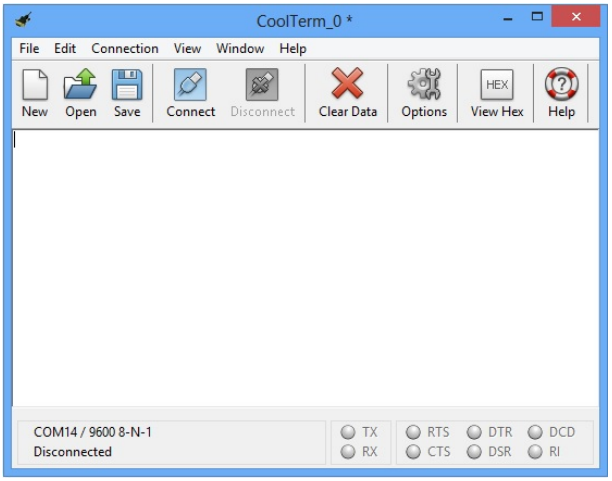


Figura 19.6.0.2 – Vista del terminal Coolterm

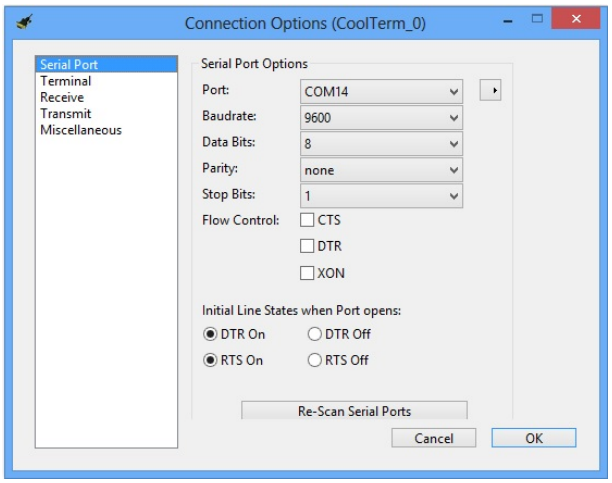


Figura 19.6.0.3 – Configuración del terminal Coolterm 1

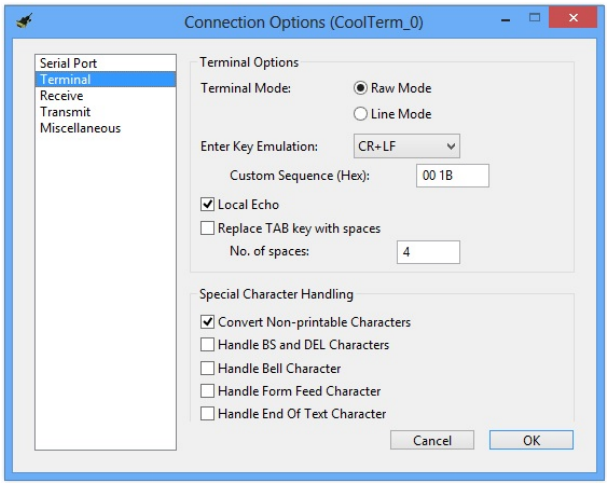


Figura 19.6.0.4 – Configuración del terminal Coolterm 2

A continuación se deben enviar los siguientes comandos por orden. Las funciones de di-

chos comandos han sido explicadas en la sección 10.Dichos comandos se detallan a continuación:

```
AT
OK
AT+CPIN=XXXX
OK
AT+CREG?
+CREG: 0,1
AT+COPS?
+COPS: 0,0,"Orange"
OK
AT+CMGF=1
OK
AT+CMGS="6XXXXXXXX"
> Enviando SMS comandos AT...
> 0x1A(hex)
```

El carácter hexadecimal “0x1A” (el último comando enviado al módulo GPS GPRS GSM V3.0), indica el fin del mensaje. Inmediatamente después, el módulo procederá a su envío, y transcurridos unos segundos el SMS enviado llegará a nuestro terminal móvil, con el texto “Enviando SMS comandos AT...”.

Todavía no hemos explicado la forma de enviar este carácter hexadecimal desde Coolterm, es muy sencillo, solo debemos seguir los siguientes pasos:

1. Abrimos la pestaña “Connection” situada en la barra superior y seleccionamos la opción “Send string”,nos abrirá una ventana donde se nos muestran 2 opciones.

2. Seleccionamos la opción Hex, entonces escribiremos el texto hexadecimal que deseamos enviar, en este caso “1A” y pulsamos el botón “Send”

3. El entorno de envío del carácter hexadecimal se muestra en la figura19.6.0.5.

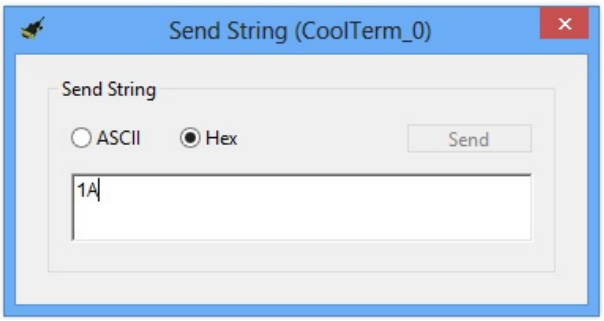


Figura 19.6.0.5 – Envío de un carácter hexadecimal utilizando el terminal Coolterm

Como todo ha salido según lo esperado y hemos recibido correctamente el SMS en nuestro terminal , pasamos ahora a hacer la prueba del envío de un SMS pero esta vez, a través de Arduino, sin necesidad de teclear manualmente los comandos AT.

19.7. Enviar SMS mediante Arduino

El presente programa envía el mensaje “Hola soy Arduino” almacenado en el array “mensaje” al número de teléfono almacenado en el array “numero_Tlf”. Los comandos AT enviados al módulo se encuentran en la función “establecer_numero_tlf()”.

El montaje para la realización de este apartado es mucho más simple de lo que se refleja en el plano [1], dado que no necesitamos conectar el LCD ni ningún pulsador adicional, solo insertar el módulo GPS en nuestro Arduino, ya sea UNO o MEGA y realizar las modificaciones pertinentes en el módulo GPS tal y como se indica en la sección 19.4.

Se recomienda seguir los pasos que se indican en el mismo para la correcta utilización del módulo.

```
1  //*************************************************************
2  // GPS/GPRS/GSM Module V3.0
3  //*************************************************************
4  // Autor : CRISTÓBAL GARCÍA CAMOIRA
5  //*************************************************************
6  // # Descripción:
7  // # El presente programa envía el mensaje "Hola soy Arduino"
8  // # almacenado en el array mensaje al número de teléfono almacenado
9  // # en el array numero_Tlf. Los comandos AT enviados al módulo se
10 // # encuentran en la función establecer_numero_tlf().
11 // # Pasos:
12 // # 1. Poner el interruptor S1 en modo Prog (hacia la derecha).
13 // # 2. poner el interruptor S2 en modo USB (hacia la izquierda).
14 // # 3. Poner el interruptor UART en la posición central.
15 // # 4. Subir el programa de Arduino a la placa.
16 // # 5. Poner el interruptor S1 en modo Com (hacia la izquierda).
17 // # 6. Poner el interruptor S2 en modo Arduino (hacia la derecha).
18 // # 7. Resetear la placa presionando el botón durante 3 segundos.
19 // # wiki link- http://www.dfrobot.com/wiki/index.php/GPS/GPRS/GSM_Module_V3.0_(SKU:
       TEL0051)
20
21 #define habilitar_gps()       digitalWrite (4, LOW)
22 #define deshabilitar_gps()    digitalWrite (4, HIGH)
23
24 #define habilitar_gsm()       digitalWrite (6, LOW)
25 #define deshabilitar_gsm()    digitalWrite (6, HIGH)
26
27 char mensaje[35] ="Hola soy Arduino";
28 char numero_Tlf[12]="600836200";
29 char numero_buf[25];
30 void setup()
31 {
32   //Inicializa los pines de control.
33   gps_inicio_pines () ;
34   digitalWrite(5,HIGH);
35   delay(1500);
```

```
36    digitalWrite(5,LOW);
37    habilitar_gsm() ;  // Habilita el modo GSM.
38    deshabilitar_gps();// Deshabilita el modo GPS.
39    delay(2000);
40    Serial.begin(9600); // Establece la velocidad de transmisión del p.serie.
41    delay(5000);
42    delay(5000);
43    delay(5000);
44  }
45
46  void loop()
47  {
48    establecer_numero_tlf (numero_Tlf);
49    envia_mensaje_gsm (mensaje);
50    finaliza_envio_gsm ();
51    while(1);
52  }
53
54  // Establece el número de teléfono.
55
56  void establecer_numero_tlf (char *numero)
57  {
58    sprintf (numero_buf, "AT+CMGS=\"%s\"", numero);
59    habilitar_gsm() ;//Habilita el modo GSM.
60    deshabilitar_gps();//Deshabilita el modo GPS.
61    Serial.println ("AT");
62    delay (2000);
63    Serial.println ("AT");
64    delay (2000);
65    Serial.println("AT+CPIN=\"5856\"");
66    delay(2000);
67    Serial.println("AT+COPS?");
68    delay(2000);
69    Serial.println ("AT+CMGF=1");
70    delay (1000);
71    Serial.println (numero_buf);
72    delay (1000);
73  }
74  // Envía el mensaje al teléfono móvil.
75  void envia_mensaje_gsm (char *mensaje)
76  {
77    Serial.println (mensaje);
78  }
79  //
80  void finaliza_envio_gsm()
81  {
82    delay (1000);
83    Serial.write (26);
84    delay (2000);
```

```
85    deshabilitar_gsm() ;// Deshabilita el modo GSM.
86    habilitar_gps();     // Habilita el modo GPS.
87  }
88
89  /*
90   Los pines por defecto para el modo GSM y GPS
91   si se desea cambiar los pines de control o
92   bien tenemos conflictos para conectar el módulo
93   en esos pines de Arduino , podremos extraer los
94   jumpers J10~J12 y unir los pines adyacentes al
95   borde de la placa a otros pines digitales.
96   */
97  // Establece los pines de trabajo del módulo GPS
98  void gps_inicio_pines ()
99  {
100     pinMode (6, OUTPUT);
101     pinMode (5, OUTPUT);
102     pinMode (4, OUTPUT);
103 }
104 //*************************************************************
```

19.8. Leer SMS almacenados en la tarjeta SIM mediante comandos AT

El programa a cargar en Arduino es idéntico al que se refleja en la sección 19.6, y por tanto solo se mostrarán al igual que en el apartado anterior los comandos AT que necesitan ser enviados al módulo para poder realizar la acción citada en dicha sección.

El montaje para la realización de este apartado es el mismo que se indica en la sección 19.6.

A diferencia que en la sección 19.6, no necesitamos mandar ningún terminador de línea en hexadecimal por el puerto serie, por tanto podremos utilizar el monitor serial de Arduino, para la comunicación serie mediante comandos “AT”con el módulo GPS.

Los pasos para la utilización del monitor serie de Arduino son los siguientes:

1. Abrimos el “Monitor Serial”

2. Vamos a la pestaña de selección de caracteres de fin de línea, y presionamos sobre la opción “Ambos NL y CR”, tal y como se muestra en la figura 19.8.0.6.Si no configuramos esto correctamente, las respuestas del módulo ante los comandos AT no se mostrarán por pantalla.

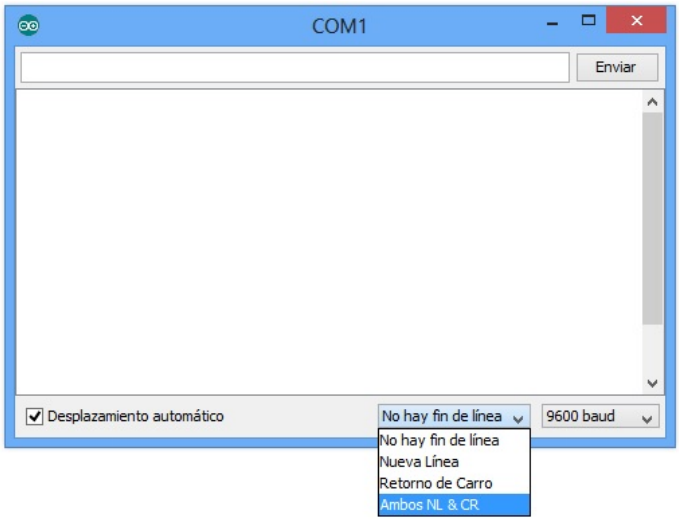


Figura 19.8.0.6 – Configuración puerto serie para enviar comandos AT

```
AT
OK
AT+CPIN=XXXX
OK
AT+COPS?
+COPS: 0,0,"Orange"
OK
AT+CMGF=1
OK
AT+CPMS="SM"
+CPMS: 6,40,6,40,6,40
OK
AT+CMGR=5
+CMGR: "REC READ","+346XXXXXXXX","","15/05/08,13:25:20+08"
Ola
OK
AT+CMGR=6
+CMGR: "REC READ","+346XXXXXXXX","","15/05/08,13:25:33+08"
Ola
OK
AT+CMGD=1,4
OK
AT+CPMS="SM"
+CPMS: 0,40,0,40,0,40
OK
AT+CMGR=1
OK
```

Comentar que la mayoría, por no decir todas, de las tarjetas SIM poseen un espacio reservado para almacenar los SMS recibidos, dependiendo del chip que inserte la tarjeta nos permitirá almacenar una mayor o menor cantidad de SMSs, en este caso estamos leyendo la quinta y sexta posición de cuarenta que nuestro chip tiene reservadas para el almacenamiento

de SMSs recibidos.

Las posiciones iniciales de nuestras tarjetas SIM recién adquiridas suelen venir con datos basura, un conjunto de números aleatorios que imagino que se grabarían de prueba para verificar el funcionamiento de la memoria del chip ,siendo las tes primeras posiciones ocupadas por estos.Aunque estos datos nos pueden servir para realizar una prueba de lectura de nuestra tarjeta, es recomendable enviar un par de SMSs desde nuestro terminal móvil durante el proceso de lectura y repetir los comandos de lectura para visualizar por consola lo que hemos enviado.

Si durante el proceso recibimos un SMS nuevo, la consola nos mostrará el siguiente código citado entre paréntesis (+CMTI: “SM”, 1), ello indica que el módulo GPS ha recibido un SMS y el número de detrás de la coma indica la posición en la que ha sido guardado.

19.9. Control de Arduino mediante SMS

El presente programa nos muestra cómo controlar Arduino mediante el envío de SMSs desde nuestro terminal móvil hacia nuestro módulo GPS/Gprs/GSM V3.0.

Este programa ha sido testado con Arduino MEGA pero es perfectamente compatible con Arduino UNO.

El montaje para la realización de este apartado es mucho más simple de lo que se refleja en el plano [1], dado que no necesitamos conectar el LCD ni ningún pulsador adicional, solo insertar el módulo GPS en nuestro Arduino, ya sea UNO o MEGA y realizar las modificaciones pertinentes en el mismo tal y como se indica en la sección 19.4, además de lo anterior se necesita conectar un diodo LED entre el pin 13 y GND.

En este ejemplo se controla el encendido y el apagado de un LED conectado al pin 13 de nuestro Arduino.

En el fragmento de código que a continuación se adjunta, se lee continuamente la primera posición de la memoria reservada para almacenamiento de SMSs de nuestra tarjeta SIM que, aunque inicialmente en el programa se borran todas las posiciones de memoria reservadas para el almacenamiento de los SMSs de nuestra tarjeta SIM en la función “setup()”, preferiblemente ha de ser borrada (mediante comandos AT).

Si no existe ningún SMS almacenado, el comando de lectura enviado por Arduino al módulo recibirá una respuesta de error.

Si existe un mensaje almacenado, Arduino procederá a su lectura.

Los SMSs que tendremos que enviarle serán “OP” para el encendido del LED y “OL” para su apagado, a cualquier otro SMS enviado desde nuestro terminal móvil cuyos contenidos sean distintos a los que hemos citado anteriormente (“OP” y “OL”) no le hará caso.

Se debe introducir en el programa el pin de nuestra tarjeta para que todo funcione correctamente, en el caso de que nos hayamos equivocado, la tarjeta SIM, tras tres intentos fallidos de introducción del PIN, se bloqueará.

La sección de código en la cual se ha de introducir el PIN se encuentra en la función

"leerSmsSim()"y es la siguiente:

```
EnviaComandoAT("AT+CPIN=5856", "OK", 2000);
```

Para desbloquearla será necesario extraerla del módulo e introducirla en nuestro terminal móvil, nuestro terminal nos mostrará en la pantalla que la tarjeta SIM está bloqueada pidiéndonos el código PUK (tendremos 10 intentos antes de que se bloquee la tarjeta definitivamente), introducimos el código PUK, y luego volveremos a introducir el código PIN, ya sea uno nuevo o el anterior, y tendremos de nuevo la tarjeta SIM desbloqueada.

En el caso de haberle enviado al módulo un SMS de forma incorrecta será necesario resetear la placa Arduino , para que vuelva a funcionar de forma correcta. Si luego de resetear la placa, el módulo GPS/GPRS/GSM conectado a nuestro Arduino sigue sin responder a los mensajes enviados desde nuestro terminal móvil, será necesario borrar todas las posiciones de memoria reservadas para el almacenamiento de los SMSs de nuestra tarjeta SIM mediante comandos "AT".

Para probar el código que se describe a continuación se recomienda, una vez cargado el programa, desconectar el cable USB que conecta nuestro Arduino con el ordenador, para evitar posibles interferencias.

```
1  //*************************************************************
2  // GPS/GPRS/GSM Module V3.0
3  //*************************************************************
4  // Autor : CRISTÓBAL GARCÍA CAMOIRA
5  //*************************************************************
6  // # Pasos:
7  // # 1. Poner el interruptor S1 en modo Prog (hacia la derecha).
8  // # 2. poner el interruptor S2 en modo USB (hacia la izquierda).
9  // # 3. Poner el interruptor UART en la posicion central.
10 // # 4. Subir el programa de Arduino a la placa.
11 // # 5. Poner el interruptor S1 en modo Com (hacia la izquierda).
12 // # 6. Poner el interruptor S2 en modo Arduino (hacia la derecha).
13 // # 7. Resetear la placa presionando el botón durante 3 segundos.
14 // # wiki link- http://www.dfrobot.com/wiki/index.php/GPS/GPRS/GSM_Module_V3.0_(SKU:
       TEL0051)
15
16   int8_t answer;
17   int x;
18   int i;
19   char SMS[100];        // Guarda toda la cadena del SMS recibido.
20
21   char *puntSms;            // Creamos un puntero de tipo char.
22   char *arrayPuntSms[5];    // y un array de punteros de tipo char.
23
24   char telefonoSms[16]; // Guarda el numero de teléfono del SMS recibido.
25   char comillas[4];     // Guarda las comillas de la segunda parte del array.
26   char fechaSms[12];    // Guarda la fecha del SMS recibido.
27   char horaSms[14];     // Guarda la hora del SMS recibido.
28   char smsRecibido[4];  // Guarda el SMS recibido.
```

```
29
30  #define habilitar_gps()      digitalWrite (4, LOW)
31  #define deshabilitar_gps()   digitalWrite (4, HIGH)
32
33  #define habilitar_gsm()      digitalWrite (6, LOW)
34  #define deshabilitar_gsm()   digitalWrite (6, HIGH)
35
36  void setup()
37  {
38   gps_inicio_pines () ;
39   pinMode (13, OUTPUT);
40   Serial.begin(9600);
41   digitalWrite(5,HIGH);
42   delay(1500);
43   digitalWrite(5,LOW);
44   delay(1500);
45   EnviaComandoAT("AT", "OK", 1000);
46   //Enciende la alimentación del GPS.
47   EnviaComandoAT("AT+CGPSPWR=1", "OK", 1000);
48   //Resetea el GPS  en modo autónomo.
49   EnviaComandoAT("AT+CGPSRST=1", "OK", 1000);
50   habilitar_gsm ();
51   deshabilitar_gps ();
52 //    Serial.println("Comenzando...");
53   EnviaComandoAT("AT", "OK", 1000);
54   EnviaComandoAT("AT+CPIN=5856", "OK", 2000);
55   EnviaComandoAT("AT+COPS?", "+COPS:", 2000);
56   EnviaComandoAT("AT+CMGD=1,4", "OK", 1000);
57   delay(2000);
58   //leerSmsSim();
59  }
60
61  void loop()
62  {
63  leerSmsSim();
64  delay(2000);
65  }
66
67  void leerSmsSim()
68 {
69  //   Serial.println("Comenzando...");
70   EnviaComandoAT("AT", "OK", 1000);
71   EnviaComandoAT("AT+CPIN=5856", "OK", 2000);
72   EnviaComandoAT("AT+COPS?", "+COPS:", 2000);
73 //    Serial.println("Modo SMS...");
74   EnviaComandoAT("AT+CMGF=1", "OK", 1000);     // Sms modo texto
75   EnviaComandoAT("AT+CPMS=\"SM\"", "OK", 1000);  // Selecciona la memoria
76   answer = EnviaComandoAT("AT+CMGR=1", "+CMGR:", 2000);  // Lee el sexto sms.
77   if (answer == 1)
```

```
78      {
79      answer = 0;
80  //    while(Serial.available() == 0);
81      // this loop reads the data of the SMS
82      do{
83      // Si hay datos en el buffer de entrada del UART, los leemos y los chequeamos
            para la respuesta.
84      if(Serial.available() > 0)
85      {
86      SMS[x] = Serial.read();
87      x++;
88
89      // Comprueba si la respuesta deseada (OK) es la respondida por el módulo.
90      if (strstr(SMS, "OK") != NULL){answer = 1;}
91      }
92      }while(answer == 0); // Esperamos por la respuesta del módulo con el time out.
93      SMS[x] = '\0';
94      Serial.print(SMS);
95      i=0;
96      x=0;
97      do
98      {
99      if (SMS[i]=='\r'){SMS[i]=',';}
100     if (SMS[i]=='\n'){SMS[i]=' ';}
101     i++;
102     }while(i<=101);
103     descomponerSms();
104     if (strcmp(smsRecibido," OP")==0)
105     {
106     digitalWrite(13,HIGH);
107     EnviaComandoAT("AT+CMGD=1,4", "OK", 1000);
108     }
109     else if (strcmp(smsRecibido," OL")==0)
110     {
111     digitalWrite(13,LOW);
112     EnviaComandoAT("AT+CMGD=1,4", "OK", 1000);
113     }
114     }
115     else
116     {
117     Serial.print("error ");
118     Serial.println(answer, DEC);
119     }
120 }
121
122  void descomponerSms()
123  {
124   i=0;
125   puntSms = strtok (SMS,","); // Tokenizamos (troceamos) la cadena que tenemos en
```

```
      el array TramaGPG por las comas
126                                 // y el primer intervalo lo guardamos en pch (puntero
                                        char)
127 //  if (strcmp(puntSms," \"REC UNREAD\"")==0)   // Si es la correcta, seguimos
      adelante
128 //  {
129   while (puntSms != NULL)        // Mientras el dato sea válido, lo aceptamos para
        llenar el array GGA
130   {
131   puntSms = strtok (NULL, ","); // Pasamos al siguiente intervalo cortado de la
        cadena
132   arrayPuntSms[i]=puntSms;                    // Guardamos el valor de puntSms en el
        array GGA
133   i++;                          // Incrementamos el contador/acumulador
134   }
135   strncpy(telefonoSms,arrayPuntSms[0] , 14); // Copia 14 chars entre las posiciones
        0-13.
136   strncpy(comillas, arrayPuntSms[1], 2);      // Copia 2 chars entre las posiciones
        0-1.
137   strncpy(fechaSms, arrayPuntSms[2], 9);      // Copia 9 chars entre las posiciones
        0-8.
138   strncpy(horaSms, arrayPuntSms[3], 12);      // Copia 12 chars entre las posiciones
        0-11.
139   strncpy(smsRecibido, arrayPuntSms[4], 3);  // Copia 3 chars entre las posiciones
        0-2.
140 //  }
141  }
142
143  int8_t EnviaComandoAT(char* comandoAT, char* resp_esperada, unsigned int timeout)
144  {
145   uint8_t x=0,  answer=0;
146   char respuesta[100];
147   unsigned long previous;
148   memset(respuesta, '\0', 100);  // Inicializa el string.
149   delay(100);
150   while( Serial.available() > 0) Serial.read();    // Limpia el buffer de entrada.
151   Serial.println(comandoAT);     // Envía el comando AT.
152   x = 0;
153   previous = millis();
154   // Este bucle espera por la respuesta.
155   do{
156   // Si hay datos en el buffer de entrada del UART, los leemos y los chequeamos para
        la respuesta.
157   if(Serial.available() != 0){
158   respuesta[x] = Serial.read();
159   x++;
160   // Comprueba si la respuesta esperada es la respuesta del módulo.
161   if (strstr(respuesta, resp_esperada) != NULL)
162   {
```

```
163    answer = 1;
164    }
165    }
166    // Espera por la respuesta con el time out
167    }while((answer == 0) && ((millis() - previous) < timeout));
168    return answer;
169  }
170
171  /*
172  Los pines por defecto para el modo GSM y GPS
173  si se desea cambiar los pines de control o
174  bien tenemos conflictos para conectar el módulo
175  en esos pines de Arduino , podremos extraer los
176  jumpers J10~J12 y unir los pines adyacentes al
177  borde de la placa a otros pines digitales.
178  */
179 // Establece los pines de trabajo del módulo GPS
180 void gps_inicio_pines ()
181 {
182 //   pinMode (3, OUTPUT);
183      pinMode (6, OUTPUT);
184      pinMode (4, OUTPUT);
185      pinMode (5, OUTPUT);
186 }
187 //*************************************************************
```

Capítulo 20

Condiciones de almacenamiento

El equipo debe estar protegido de la luz solar para evitar daños en el LCD. La temperatura de almacenamiento debe ser inferior a 35°C. El ambiente debe ser seco y con ausencia de polvo.

Capítulo 21

Guía de implementación

En esta última parte del pliego se elaboran las condiciones de implementación del equipo. Cuando se pretenda realizar el montaje del mismo se deberá recurrir a esta guía para asegurar que el montaje de los dispositivos sea el correcto.

21.1. Primeros pasos

Antes de comenzar a programar nuestro Arduino se deben instalar los drivers del mismo y el software de programación adecuados. Esto se puede descargar desde la página oficial de Arduino, en el siguiente enlace:

- `http://arduino.cc/en/Main/OldSoftwareReleases`

En este enlace hay que seleccionar el paquete disponible acorde al sistema operativo que se tenga instalado en el ordenador donde se va a usar el Arduino. Se recomienda descargar la versión de Arduino IDE 1.0.5 , para ello, hacemos click donde se muestra Windows Installer, tal y como muestra la figura 21.1.0.1. Este archivo es un instalador automático del compilador de Arduino, solo hay que seguir los pasos que indicaremos a continuación, aunque son bastante intuitivos, los explicaremos para que no queden dudas. De esta forma se asegura la compatibilidad de todas las librerías utilizadas durante la fase de realización del proyecto.

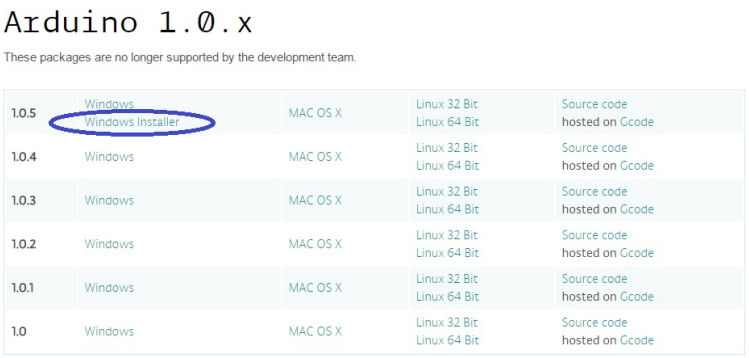


Figura 21.1.0.1 – Instalación del IDE Arduino 1

Una vez descargado el fichero ejecutable del enlace anterior presionamos en el recuadro donde se muestra **I agree** tal y como se muestra en la figura 21.1.0.2:

Figura 21.1.0.2 – Instalación del IDE Arduino 2

Seleccionamos todos los componentes que se indican, como se muestra en la figura 21.1.0.3 y presionamos **Next**.

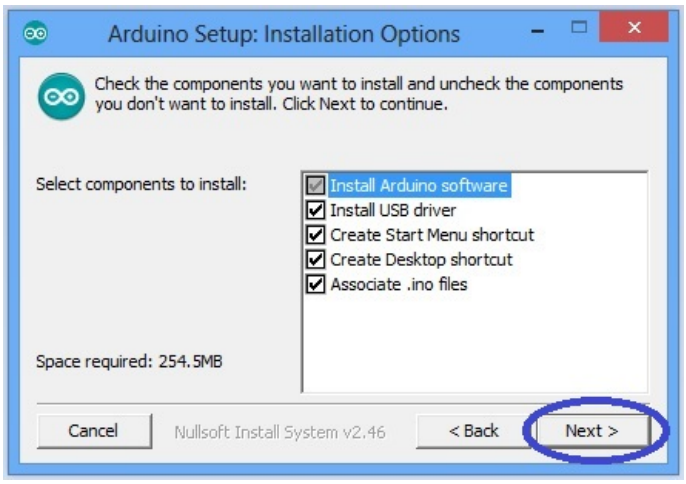


Figura 21.1.0.3 – Instalación del IDE Arduino 3

El siguiente paso es indicarle la ruta de instalación de Arduino, ruta que por defecto es la siguiente:

```
1 C:\Program Files (x86)\Arduino
```

El sistema operativo utilizado durante la instalación ha sido Windows 8, puede que en Windows Vista, 7, o XP venga determinada por defecto una ruta diferente.

Esperamos unos minutos y ya tendremos instalado el compilador de Arduino.

Los drivers que controlan las diferentes placas de Arduino se instalaran automáticamente en Windows 8, solo tendremos que conectar Arduino a nuestro PC y en la pantalla nos aparecerá un ventana como la que que se muestra en la figura 21.1.0.4.

El sistema operativo se encargará del resto. En esta parte de la instalación es necesario tener un poco mas de paciencia.

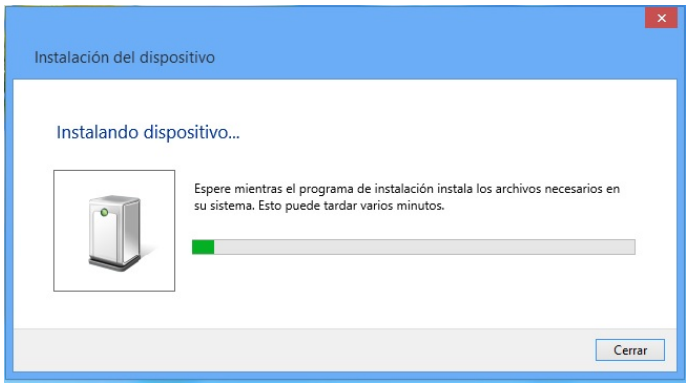


Figura 21.1.0.4 – Instalación del IDE Arduino 4

Una vez instalado el programa y los drivers, y con el Arduino ya conectado, necesitamos saber qué puerto COM le ha asignado nuestro PC, para ello hacemos click derecho en Equipo, clickeamos en Administrar – Administrador de dispositivos – Puertos (COM y LPT) y veremos qué puerto le ha asignado nuestro computador. Esto se muestra en la figura 21.1.0.5.

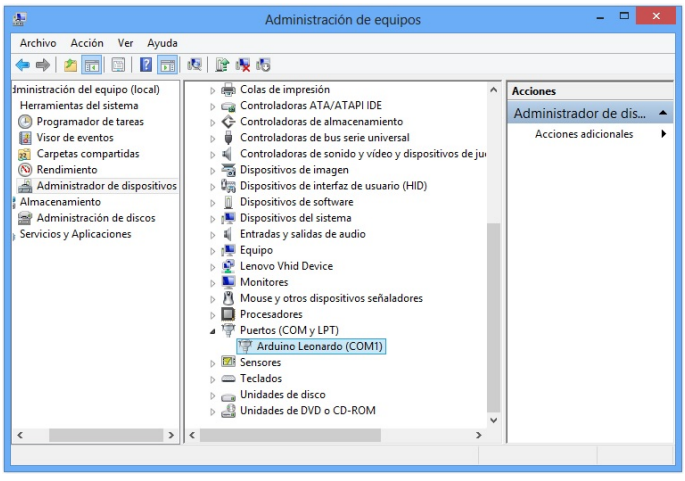


Figura 21.1.0.5 – Instalación del IDE Arduino 5

En el caso de que tengamos otro sistema operativo tal y como windows XP y los drivers no se nos instalan automáticamente, podremos seleccionarlos de la siguiente carpeta situada en el directorio en el cual hemos instalado el programa de Arduino.

Esta carpeta se muestra en la figura 21.1.0.6.

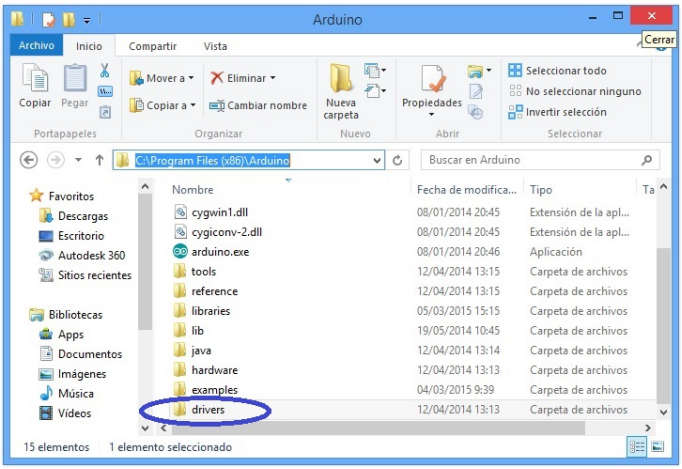


Figura 21.1.0.6 – Instalación del IDE Arduino 6

Una vez en este punto ya podremos programar nuestro Arduino con el código fuente necesario . No es necesario instalar librerías adicionales a nuestro software , con las que trae por defecto el paquete que hemos instalado es suficiente.

Una vez hecho esto el software y el hardware de Arduino están preparados para ser programados de forma que los periféricos conectados funcionen de forma correcta y con una programación sencilla y muy intuitiva. Lo que se debe hacer a continuación es compilar y cargar el archivo fuente.ino con todo el código fuente en el Arduino Mega.

21.2. Montaje de los circuitos

El montaje de nuestro equipo completo se puede subdividir en dos partes, y es que fundamentalmente son dos los circuitos que conforman nuestro montaje, tres , si tenemos en cuenta un tercer circuito realizado, aunque forma parte de uno de los anteriores, estos circuitos son:

1. Circuito del Arduino y el módulo GPS/GPRS/GSM V3.0. Este circuito se corresponde con el del plano [1].

2. Circuito de la placa de control del LCD . El diseño de este circuito lo podemos encontrar en el plano [3].

3. Circuito de la placa de inserción del encoder rotativo. El diseño de este circuito lo podemos encontrar en el plano [3].

4. Modificación de los pines de control por defecto del módulo GPS/GPRS GSM. En la sección 19.4 encontraremos toda la información pertinente a este proceso con los pasos muy detallados.

El montaje del primer circuito es el más sencillo de todos, el circuito que se indica en el plano [1] (fijándonos en las conexiones del módulo GPS y el Arduino Mega) es el que más

líneas de uniones tiene, esto es debido a que el módulo GPS va insertado encima del Arduino Mega, coincidiendo los pines del Arduino Mega con los pines del módulo GPS.

El módulo GPS va insertado a presión sobre la placa de Arcuino Mega.La figura 21.2.0.7 muestra la inserción de ambas placas.

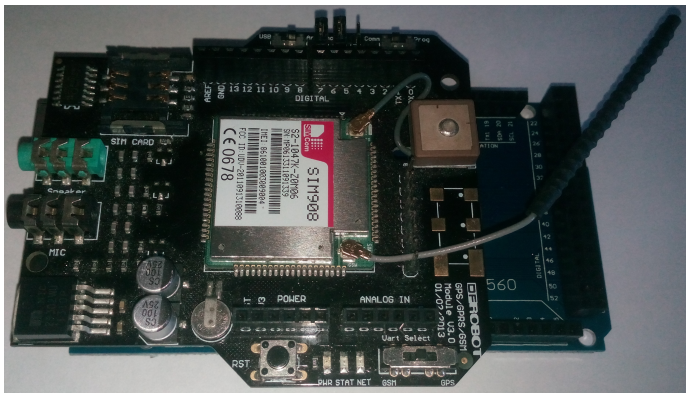


Figura 21.2.0.7 – Inserción del módulo GPS sobre la placa Arduino

Para la realización del montaje del segundo circuito, un tanto más laborioso que el primero, se ha utilizado una placa fotosensible de doble cara , cuyos fotolitos podemos encontrar en el plano [7] y en el plano[5] tomando como referencia el diseño del plano [1], también poseemos el plano de situación de los componentes (plano [6]).

El plano de disposición de los componentes no es muy claro, dado que existen componentes colocados por ambas partes de la placa y éstos se superponen, por ello, presentamos una imagen de la placa resuelta, esta se corresponde con la de la figura 21.2.0.8.

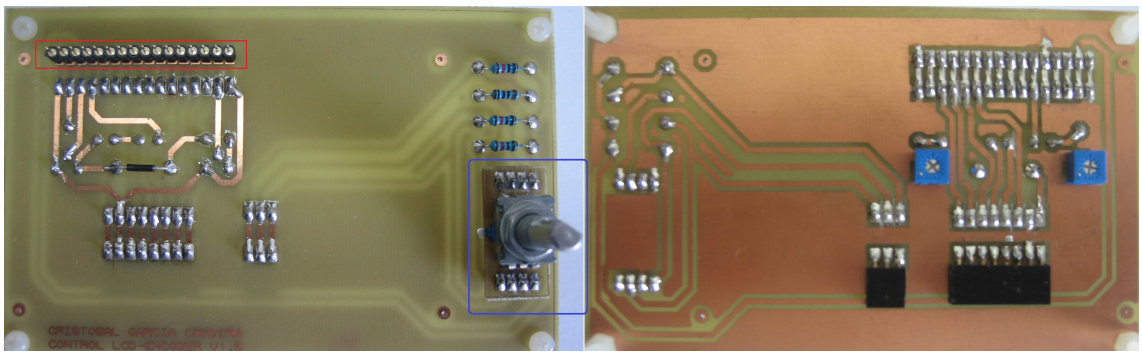


Figura 21.2.0.8 – Placa de control del LCD

NOTA:El LCD se inserta en los pines enmarcados con un recuadro rojo, indicados en la figura anterior (21.2.0.8).

Para la realización del montaje del tercer circuito,la placa de inserción del encoder rotativo, que ya se presenta insertado en la placa de control del LCD en la figura 21.2.0.8 enmarcado en un recuadro de color azul se han tenido en cuenta dos planos: el de diseño (plano [7]) y el de su fotolito para la elaboración de su placa (plano [8]).

Los componentes mencionados en el Estado de Mediciones son los que conforman las placas de circuito impreso diseñadas. Los planos (plano [4] y plano [8]) están a escala real. De esta forma se pueden imprimir directamente en papel transparente para poder insolarlos posteriormente.

El proceso de fabricación de las placas de circuito impreso se realiza por el método de la insolación, este proceso es un proceso sencillo y que se puede consultar fácilmente en la web.

21.3. Conexionado de los circuitos

Se debe consultar el plano general de montaje (plano [1]) para realizar la conexión de los distintos circuitos implementados anteriormente.

Cabe comentar en este apartado que el motivo de la realización de la placa de control del LCD ha sido para facilitar y simplificar el conexionado de nuestro Arduino con el módulo GPS, esta simplicidad en el montaje nos ha ahorrado mucho tiempo a la hora de montar y desmontar el LCD en la fase de pruebas.

En el plano [6], no podemos ver a qué pines de nuestro Arduino están insertadas las conexiones procedentes de la Placa de control del LCD, por tanto la figura 21.3.0.9 y los comentarios posteriores nos aclararán su conexionado.

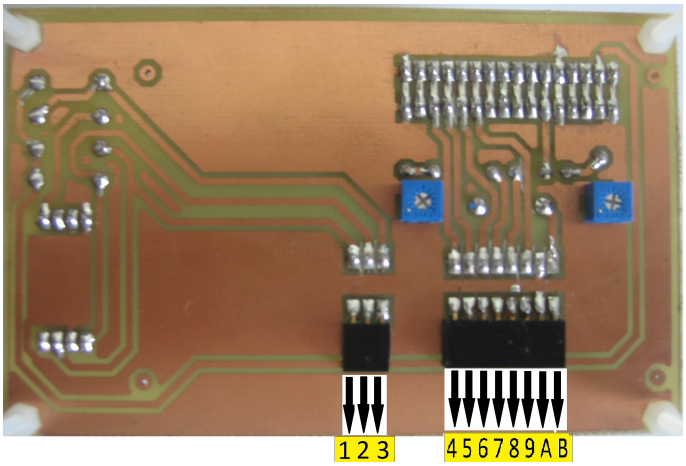


Figura 21.3.0.9 – Conexión a Arduino de la placa de control del LCD

- Pin 1 de la figura 21.3.0.9 al pin 2 de nuestro Arduino.
- Pin 2 de la figura 21.3.0.9 al pin 3 de nuestro Arduino.
- Pin 3 de la figura 21.3.0.9 al pin 19 de nuestro Arduino MEGA.Pin 7 en UNO.
- Pin 4 de la figura 21.3.0.9 al pin 8 de nuestro Arduino.
- Pin 5 de la figura 21.3.0.9 al pin 9 de nuestro Arduino.
- Pin 6 de la figura 21.3.0.9 al pin 10 de nuestro Arduino.

- Pin 7 de la figura 21.3.0.9 al pin 11 de nuestro Arduino.
- Pin 8 de la figura 21.3.0.9 al pin 12 de nuestro Arduino.
- Pin 9 de la figura 21.3.0.9 al pin 13 de nuestro Arduino.
- Pin A de la figura 21.3.0.9 al pin GND de nuestro Arduino.
- Pin B de la figura 21.3.0.9 al pin 5V de nuestro Arduino.

21.4. Alimentación del circuito

La alimentación ideal para nuestro dispositivo es de 9V y 2200mA.

En el mercado existen cargadores muy baratos que nos ofrecen las características anteriores , aunque ello nos limita el uso del localizador a espacios cerrados donde exista una toma de corriente.

En la práctica y en primer lugar, para la alimentación del circuito, se utilizó una batería de 9 voltios y 200mA,corriente escasa pero suficiente para poder funcionar sin enviar ningún SMS .

En segundo lugar , se utilizó la fuente de alimentación de un ordenador, que posee, entre otras muchas tensiones de salida, 12V y corriente suficiente, dado que dicha fuente es capaz de alimentar un ordenador de sobremesa con un gasto de hasta 400W,lo que ocurre es que el regulador de tensión (MIC29302WU) que posee el módulo GPS GPRS GSM, se calienta a pesar de las características eléctricas que este posee, esto es debido a que la disposición de los componentes del circuito que alimenta el módulo SIM908, del cual dispone dicho módulo GPS, esta preparado para admitir una tensión de 5V y ser regulada a 3,7V este circuito se observa en la figura 7.0.0.1. Por otro lado, también estamos admitiendo que estamos en el límite superior de voltaje recomendado por el fabricante para alimentar nuestro Arduino, este recomienda una tensión de entre 6 y 12 V,por tanto esta segunda opción no es muy recomendable para la alimentación del dispositivo durante largos periodos de tiempo.

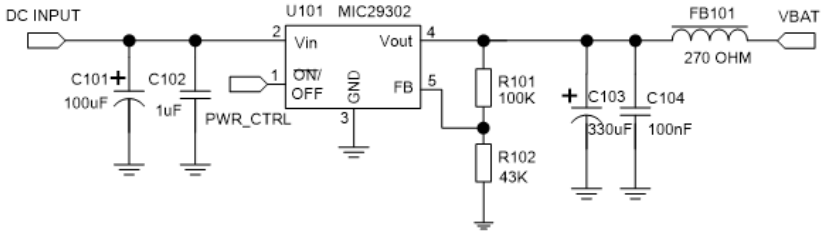


Figura 21.4.0.10 – Circuito de referencia para la alimentación del módulo

Por último y la mejor alternativa es utilizar una batería lipo comercial recargable de 7,4 V y 2200 mA, de esta forma ya sí tendríamos la energía necesaria para afrontar el gasto que conlleva el envío de un SMS.

En el apartado del manual de usuario y especificaciones se ha comentado que en momentos puntuales,durante el envío de un SMS,se pueden alcanzar corrientes de hasta 2 A.

La batería sólo alimentará el Arduino ya que el resto de tensiones se obtienen de él.

21.5. Montaje final en caja

Una vez conexionado todo el equipo, se debe realizar su montaje en una caja. Los distintos circuitos irán atornillados a ella de forma que queden fijos.

La caja posee un agujero para pasar el cable de alimentación del equipo hacia el interior de la caja. Además deberá permitir acceder al puerto USB del Arduino al exterior de la caja.

Se debe asegurar de que todas las masas estén conectadas de forma que no haya ningún punto flotante. El display LCD se montará en la parte superior de la caja. El display irá fijado a la caja por medio de cuatro tornillos, uno en cada una de las esquinas.

La caja en cuestión fue diseñada con Solid Edge y ha sido ejecutada en una impresora 3D, las referencias de los planos de la misma son el plano(10) el plano (11) y el plano (12).

A continuación mostraremos unas imágenes acerca del resultado final de la caja en cuestión , una imagen en la que se muestra la base de la caja durante el proceso de fabricación (figura 21.5.0.11), la base de la caja (figura 21.5.0.12) como de la tapa superior que sujeta el LCD (figura 21.5.0.13).

Figura 21.5.0.11 – Imagen de la base de la caja del localizador en proceso de fabricación

Figura 21.5.0.12 – Imagen de la base de la caja del localizador

Figura 21.5.0.13 – Imagen de la tapa de la caja del localizador

TÍTULO: **CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO**

ESTADO DE MEDICIONES

PETICIONARIO: **ESCOLA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2015

AUTOR: EL ALUMNO

Fdo.: CRISTÓBAL GARCÍA CAMOIRA

Índice del Estado de Mediciones

Capítulo 22

Dispositivos del equipo

ID	Imagen	Descripción	Ref. RS	Ref.Farnell	Ud
SIM1		Arduino Mega 2560 REV 3 A000067	715-4084	2212779	1
LCD1		Display LCD 20x4 LCD 2004	720-0226	2342648	1
GPS		Módulo GPS/GPRS/GSM V3.0	————	————	1
BAT		Batería Lipo 7,4V 2200mAH	————	————	1

Tabla 22.0.0.1 – Lista de materiales 1

Capítulo 23

Placa de control del LCD

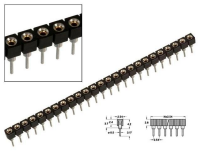
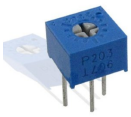
ID	Imagen	Descripción	Ref. RS	Ref.Farnell	Ud
ARD		Tira de pines hembra	766-6679	1593469	1
RENC1		Encoder rotativo	729-5684	2065021	1
R4 R2 R3		Resistencia 10 K	707-7745	1265079	3
R1		Resistencia 330	135-308	1128021	1
RV1,RV2		Potenciómetro (trimmer Cermet) 10K	522-2227	9354301	2

Tabla 23.0.0.1 – Lista de materiales 2

—		Placa fotosensible de doble cara (16x10 cm)	159-6108	1267741	1

Tabla 23.0.0.2 – Lista de materiales 3

Capítulo 24

Tarjeta de comunicaciones y terminal móvil

ID	Imagen	Descripción	Ref. RS	Ref.Farnell	Ud
TARJETA SIM		Tarjeta SIM	————	————	1
TERM MOVIL		Terminal móvil	————	————	1

Tabla 24.0.0.1 – Lista de materiales 4

Tanto la tarjeta de comunicaciones (Tarjeta SIM),como el componente GPS y el terminal móvil, no se encuentran en las tiendas online Farnell ni RS, en el caso de la tarjeta de comunicaciones, seria necesario informarnos con la compañía que nos ofrezca una mejor oferta, y en el caso del gps utilizado en el presente proyecto, podremos realizar la compra de forma online desde el siguiente enlace web:

- `http://www.dfrobot.com/index.php?route=product/product&product_id=673`

TÍTULO: **CONTROL DE POSICIONAMIENTO DE VEHÍCULOS MEDIANTE ARDUINO**

PRESUPUESTO

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2015

AUTOR: EL ALUMNO

Fdo.: CRISTÓBAL GARCÍA CAMOIRA

Índice del presupuesto

Capítulo 25

Presupuesto de materiales

25.1. Presupuesto de dispositivos

ID	Imagen	Descripción	PVP RS	PVP Farnell	Ud	Total
SIM1		Arduino Mega 2560 REV 3 A000067	38,76 €	35,81 €	1	35,81 €
LCD1		Display LCD 20x4 LCD 2004	11,94 €	14,42 €	1	11,94 €
GPS		Módulo GPS/GPRS/GSM V3.0	—	—	1	89,50 €
BAT		Batería Lipo 7,4V 2200mAH	—	—	1	22,5 €
					Total:	159,75 €

Tabla 25.1.0.1 – Presupuesto de dispositivos

Tanto la tarjeta SIM como el terminal móvil no se incluyen en el presupuesto dado que su precio es variable, además presuponemos que ya se dispone de estos dos artículos.

25.2. Presupuesto de la placa de control del LCD

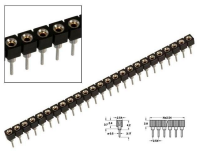
ID	Imagen	Descripción	PVP RS	PVP Farnell	Ud	Total
ARD		Tira de pines hembra	2,88 €	1,73 €	1	3,46 €
RENC1		Encoder rotativo	9,15 €	3,86 €	1	3,86 €
R4 R2 R3		Resistencia 10 K	0,015 €	0,142 €	3	0,05 €
R1		Resistencia 330	0,036 €	0,009 €	1	0,02 €
RV1,RV2		Potenciómetro (trimmer Cermet) 10K	1,63 €	0,926 €	2	1,86 €
——-		Placa fotosensible de doble cara (16x10 cm)	5,30 €	5,51 €	1	5,30 €
					Total:	14,54 €

Tabla 25.2.0.2 – Presupuesto placa de control LCD

Capítulo 26

Presupuesto de material de montaje

Concepto	Imagen	Cantidad	Precio ud	Total
Taladro Proxxon y accesorios		1	70€	70 €
Soporte Proxxon		1	60€	60 €
Tenazas electrónicas		1	5 €	5 €
Alicates electrónicos		1	5 €	5 €
Polímetro		1	20 €	20 €

Tabla 26.0.0.1 – Presupuesto de material de montaje 1

Concepto	Imagen	Cantidad	Precio ud	Total
Soldador		1	30 €	30 €
Estaño		1	5 €	5 €
			Total:	195 €

Tabla 26.0.0.2 – Presupuesto de material de montaje 2

Capítulo 27

Presupuesto de Recursos humanos

Concepto	Nº de horas	Precio/hora	TOTAL
Ingeniería	300	60 €	18000 €
Montaje	18	40 €	720 €
		Total:	18720 €

Tabla 27.0.0.1 – Presupuesto de recursos humanos

Capítulo 28

Presupuesto final

Concepto	SUBTOTAL
Ingeniería	18000 €
Montaje	720 €
Dispositivos	159,75 €
Placa de control LCD	14,54 €
Material de montaje	195 €
IVA (21)	4008,75 €
TOTAL:	23098,04 €

Tabla 28.0.0.1 – Presupuesto final

El presupuesto total del proyecto asciende a la cantidad de: ***Veintitrés mil noventa y ocho euros y cuatro céntimos*** (23098,04 €)